09/777,283 #15

The Patent Office

INVESTOR IN PEOPLE

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

Signed Stephen Hordley.

Dated 11 February 2004

PATENT OFFICE
... use
28 FEB 1995
RECEIVED BY HAND

28 FEB 1995

1MAR95 E00289229 PAT 1 77 UC 25.00
28 FEB 1995

Your reference PH/ P18525GB

9504046.5

otes
ease type, or write in dark ink
ing CAPITAL letters. A
escribed fee is payable for a
quest for grant of a patent.
r details, please contact the
atent Office (telephone
71–438 4700).

ule 16 of the Patents Rules 1990
the main rule governing the
mpletion and filing of this form.

❷ Do not give trading styles, for example, 'Trading as XYZ company', nationality or former names, for example, 'formerly (known as) ABC Ltd' as these are not required.

Warning
After an application for a Patent has been filed, the Comptroller of the Patent Office will consider whether publication or communication of the invention should be prohibited or restricted under Section 22 of the Patents Act 1977 and will inform the applicant if such prohibition or restriction is necessary. Applicants resident in the United Kingdom are also reminded that under Section 23, applications may not be filed abroad without written permission unless an application has been filed not less than 6 weeks previously in the United Kingdom for a patent for the same invention and either no direction prohibiting publication or communication has been given, or any such direction revoked.

The Patent Office

Request for grant of a Patent

Form 1/77 **Patents Act 1977**

❶ Title of invention

1 Please give the title of the invention PIPELINE

❷ Applicant's details

❑ **First or only applicant**

2a If you are applying as a corporate body please give:

Corporate name DISCOVISION ASSOCIATES

Country *(and State of incorporation, if appropriate)* UNITED STATES OF AMERICA
STATE OF CALIFORNIA

2b If you are applying as an individual or one of a partnership please give in full:

Surname

Forenames

2c **In all cases**, please give the following details:

Address

2355 Main Street, Suite 200,
Irvine, California 92714,
United States of America.

UK postcode *(if applicable)*

Country United States of America

ADP number *(if known)* 414435002 P.A.

❹ Reference numb r

4 Agent's or applicant's reference number *(if applicable)* PH/P18525GB

❺ Claiming an earli r application dat

5 Are you claiming that this application be treated as having been filed on the date of filing of an earlier application?

Please mark correct box

Yes ☐ No ☐ ➡ ***go to 6***

⬇ ***please give details below***

❑ number of earlier application or patent number

❑ filing date

(day month year)

❑ and the Section of the Patents Act 1977 under which you are claiming:

Please mark correct box

15(4) (Divisional) ☐ 8(3) ☐ 12(6) ☐ 37(4) ☐

❻ Declaration of priority

❻ If you are declaring priority from a PCT Application please enter 'PCT' as the country and enter the country code (for example, GB) as part of the application number.

Please give the date in all number format, for example, 31/05/90 for 31 May 1990.

6 If you are declaring priority from previous application(s), please give:

Country of filing	Priority application number *(if known)*	Filing date *(day, month, year)*
Great Britain	9405914.4	24.3.1994

For official use

THE PATENT OFFICE
28 FEB 1995
RECEIVED BY HAND

28 FEB 1995

Your reference PH/ P18525GB

9504046.5

Notes
Please type, or write in dark ink using CAPITAL letters.

No fee is required with this form.

Rule 15 of the Patents Rules 1990 is the main rule governing the completion and filing of this form.

If you do not have enough space please use a separate sheet of paper.

When an application does not declare priority at all, or declares priority from an earlier UK application, sufficient additional copies of this form must be supplied to enable the Comptroller to send one to each inventor who is not an applicant.

The Patent Office

Statement of inventorship and of right to grant of a Patent

Form 7/77 **Patents Act 1977**

❶ Application details

1a Please give the patent application number *(if known)*:

1b Please give the full name(s) of the applicant(s):
DISCOVISION ASSOCIATES

❷ Title of invention

2 Please give the title of the invention:
PIPELINE

❸ Derivation of right

3 Please state how the applicant(s) derive(s) the right to be granted a patent:
By virtue of the inventors employment and a subsequent assignment.

❹ Declaration

4 I believe the person(s) named overleaf (and on any supplementary copies of this form) to be the inventor(s) of the invention for which the patent application has been made. I consent to the disclosure of the details contained in this form to each inventor named.

Please sign here Signed [illegible] Date 24. 2. 1995. *(day month year)*
KILBURN & STRODE

Please turn over

DUPLICATE

PIPELINE

INTRODUCTION

The present invention is directed to improvements in methods and apparatus for decompression which operates to decompress and/or decode a plurality of differently encoded input signals. The illustrative embodiment chosen for description hereinafter relates to the decoding of a plurality of encoded picture standards. More specifically, this embodiment relates to the decoding of any one of the well known standards known as JPEG, MPEG and H.261.

A serial pipeline processing system of the present invention comprises a single two-wire bus used for carrying unique and specialized interactive interfacing tokens, in the form of control tokens and data tokens, to a plurality of adaptive decompression circuits and the like positioned as a reconfigurable pipeline processor.

PRIOR ART

One prior art system is described in United States Patent No. 5,216,724. The apparatus comprises a plurality of compute modules, in a preferred embodiment, for a total of four compute modules coupled in parallel. Each of the compute modules has a processor, dual port memory, scratch-pad memory, and an arbitration mechanism. A first bus couples the compute modules and a host processor. The device comprises a shared memory which is coupled to the host processor and to the compute modules with a second bus.

United States Patent No. 4,785,349 discloses a full motion color digital video signal that is compressed, adaptively acquired quad-tree division structure. Upon initialization of the system, a uniform, prescribed gray scale value or picture half-tone expressed as a defined luminance value is written into the image store of a coder at the transmitter and in the image store of a decoder at the receiver store, in the same way for all picture elements (pixels). Both the image store in the coder as well as the image store in the decoder are each operated with feed back to themselves in a manner such that the content of the image store in the coder and decoder can be read out in blocks of variable size, can be amplified with a factor greater than or less than 1 of the luminance and can be written back into the image store with shifted addresses, whereby the blocks of variable size are organized according to a known quad tree data structure.

United States Patent No. 5,122,875 discloses an apparatus for encoding/decoding an HDTV signal. The apparatus includes a compression circuit responsive to high definition video source signals for providing hierarchically layered codewords CW representing compressed video data and associated codewords T, defining the types of data represented by the codewords CW. A priority selection circuit, responsive to the codewords CW and T, parses the codewords CW into high and low priority codeword sequences wherein the high and low priority codeword sequences correspond to compressed video data of relatively greater and lesser importance to image reproduction respectively. A transport processor, responsive to the high and low priority codeword sequences, forms high and low priority transport blocks of high and low priority codewords, respectively. Each transport block includes a header, codewords CW and error detection check bits. The respective transport blocks are applied to a forward error check circuit for applying additional error check data. Thereafter, the high and low frequency terms is reduced, this being followed by inverse transformation to produce a reduced-size matrix of sample points representing the original block of data. In the case of interpolation, additional frequency components of zero value are inserted into the array of frequency components after which inverse transformation produces an enlarged data sampling set without an increase in spectral bandwidth. In the case of sharpening, accomplished by a convolution or filtering operation involving multiplication of transforms of data and filter kernel in the frequency domain, there is provided an inverse transformation resulting in a set of blocks of processed data samples. The blocks are overlapped followed by a savings of designated samples, and a discarding of excess samples from regions of overlap. The spatial representation of the kernel is modified by reduction of the number of components, for a linear-phase filter, and zero-padded to equal the number of samples of a data block, this being followed by forming the discrete odd cosine transform (DOCT) of the padded kernel matrix.

United States Patent No. 5,175,617 discloses a system and method for transmitting logmap video images through telephone line band-limited analog channels. The pixel organization in the logmap image is designed to match the sensor geometry of the human eye with a greater concentration of pixels at the center. The transmitter divides the frequency band into channels, and assigns one or two pixels to each channel, for example a 3KHz voice quality telephone line is divided into 768 channels spaced about 3.9Hz apart. Each channel consists of two carrier waves in quadrature, so each channel can carry two pixels. Some channels are reserved for special calibration signals enabling the receiver to detect both the phase and magnitude of the received signal. If the sensor and pixels are connected directly to a bank of oscillators and the receiver can upon.

United States Patent No. 5,231,484 discloses a system and method for implementing an encoder suitable for use with the proposed ISO/IEC MPEG standards. Included are three cooperating components or subsystems that operate to variously adaptively pre-process the incoming digital motion video sequences, allocate bits to the pictures in a sequence, and adaptively quantize transform coefficients in different regions of a picture in a video sequence so as to provide optimal visual quality given the number of bits allocated to that picture.

United States Patent No. 5,267,334 discloses a method of removing frame redundancy in a computer system for a sequence of moving images. The method comprises detecting a first scene change in the sequence of moving images and generating a first keyframe containing complete scene information for a first image. The first keyframe is known, in a preferred embodiment, as a "forward-facing" keyframe or intraframe, and it is normally present in CCITT compressed video data. The process then comprises generating at least one intermediate compressed frame, the at least one intermediate compressed frame containing difference information from the first image for at least one image following the first image in time in the sequence of moving images. This at least one frame being known as an interframe. Finally, detecting a second scene change in the sequence of moving images and generating a second keyframe containing complete scene information for an image displayed at the time just prior to the second scene change, known as a "backward-facing" keyframe. The first keyframe and the at least one intermediate compressed frame are linked for forward play, and the second keyframe and the intermediate compressed frames are linked in reverse for reverse play. The intraframe may also be used for generation of complete scene information when the images are played in compared with the accumulated desired number of bits expended, for a selected number of sectors associated with the particular group of data. The system then readjusts the quantization step size to target a final desired number of data bits for a plurality of sectors, for example describing an image. Various methods are described for updating the quantization step size and determining desired bit allocations.

The article, Chong, Yong M., A Data-Flow Architecture for Digital Image Processing, Wescon Technical Papers: No. 2 Oct./Nov. 1984, discloses a real-time signal processing system specifically designed for image processing. More particularly, a token based data-flow architecture is disclosed wherein the tokens are of a fixed one word width having a fixed width address field. The system contains a plurality of identical flow processors connected in a ring fashion. The tokens contain a data field, a control field and a tag. The tag field of the token is further broken down into a processor address field and an identifier field. The processor address field is used to direct the tokens to the correct data-flow processor, and the identifier field is used to label the data such that the data-flow processor knows what to do with the data.. In this way, the identifier field acts as an instruction for the data-flow processor. The system directs each token to a specific data-flow processor using a module number (MN). If the MN matches the MN of the particular stage, then the appropriate operations are performed upon the data. If unrecognized, the token is directed to an output data bus.

The article, Kimori, S. et al. An Elastic Pipeline Mechanism by Self-Timed Circuits, IEEE J. of Solid-State Circuits, Vol. 23, No. 1, February 1988, discloses an elastic pipeline having self-timed circuits. The asynchronous

The present invention relates to an improved pipeline system having an input, an output and a plurality of processing stages between the input and the output, th plurality of processing stages being interconnected by a two-wire interface for conveyance of tokens along the pipeline, and control and/or DATA tokens in the form of universal adaptation units for interfacing with all of the processing stages in the pipeline and interacting with selected stages in the pipeline for control data and/or combined control-data functions among the processing stages, so that the processing stages in the pipeline are afforded enhanced flexibility in configuration and processing. In accordance with the invention, the processing stages may be configurable in response to recognition of at least one token.

DESCRIPTION OF THE DRAWINGS

Figure. 1 illustrates six cycles of a six-stage pipeline for different combinations of two internal control signals;

Figures. 2a and 2b illustrate a pipeline in which each stage includes auxiliary data storage. They also show the manner in which pipeline stages can "compress" and "expand" in response to delays in the pipeline;

Figures. 3a(1), 3a(2), 3b(1) and 3b(2) illustrate the control of data transfer between stages of a preferred embodiment of a pipeline using a two-wire interface and a multi-phase clock;

Figure. 4 is a block diagram that illustrates a basic embodiment of a pipeline stage that incorporates a two-wire transfer control and also shows two consecutive pipeline processing stages with the two-wire transfer control;

Figures. 5a and 5b taken together depict one example of a timing diagram that shows the relationship between timing signals, input and output data, and internal control signals used in the pipeline stage as shown in Figure. 4;

Figure. 6 is a block diagram of one example of a pipeline stage that holds its state under the control of an extension bit;

Figure. 7 is a block diagram of a pipeline stage that decodes stage activation data words;

Figures. 8a and 8b taken together form a block diagram showing the use of the two-wire transfer control in an exemplifying "data duplication" pipeline stage;

Figures. 9a and 9b taken together depict one example of a timing diagram that shows the two-phase clock, the two-wire transfer control signals and the other internal data and control signals used in the exemplifying embodiment shown in Figures. 8a and 8b.

Figure 10 is a block diagram of a reconfigurable processing stage;

offset by (1,1);
Figure 27 is a block diagram illustrating the Huffman decoder and parser state machine of the Spatial Decoder ;
Figure 28 is a block diagram illustrating the prediction filter.

Figure 140 is a block diagram showing; microprocessor test access;
Figure 141 shows 1-D Transform Micro-Architecture;
Figure 142 shows a temporal decoder block diagram;
Figure 143 shows the structure of a Two-wire interface stage;
Figure 144 shows the address generator block diagram;
Figure 145 shows the block and pixel offsets;
Figure 146 shows multiple prediction filters;
Figure 147 shows a single prediction filter;
Figure 148 shows the 1-D prediction filter;
Figure 149 shows a block of pixels;
Figure 150 shows the structure of the read rudder;
Figure 151 shows the block and pixel offsets;
Figure 152 shows a prediction example;
Figure 153 shows the read cycle;
Figure 154 shows the write cycle;
Figure 155 shows the top-level registers block diagram with timing references;
Figure 156 shows the control for incrementing presentation numbers;
Figure 157 shows the buffer manager state machine (complete);
Figure 158 shows the state machine main loop;
Figure 159 shows the buffer 0 containing an SIF (22 by 18 macroblocks) picture;
Figure 160 shows the SIF component 0 with a display window;
Figure 161 shows an example picture format showing storage block address;
Figure 162 shows a buffer 0 containing a SIF (22 by 18 macroblocks) picture;
Figure 163 shows an example address calculation;
Figure 164 shows a write address generation state machine;
Figure 165 shows a slice of the datapath;
Figure 166 shows a two cycle operation of the datapath;
Figure 167 shows mode 1 filtering;
Figure 168 shows a horizontal up-sampler datapath; and
Figure 169 shows the structure of the color-space converter.

tokens, and the interactions with a proc ssing stag may be determined by such address fields.

In an improved pipeline machine, in accordance with the invention, the tokens may include an extension bit for each token, the extension bit indicating the presence of additional words in that token and identifying the last word in that token. The address fields may be of variable length and may also be Huffman coded.

In the improved pipeline machine, the tokens may b generated by a processing stage. Such pipeline tokens may include data for transfer to the processing stages or the tokens may be devoid of data. Some of the tokens may be identified as DATA tokens and provide data to the processing stages in the pipeline, while other tokens are identified as control tokens and only condition the processing stages in the pipeline, such conditioning including reconfiguring of the processing stages. Still other tokens may provide both data and conditioning to the processing stages in the pipeline. Some of said tokens may identify coding standards to the processing stages in the pipeline, whereas other tokens may operate independent of any coding standard among the processing stages. The tokens may be capable of successive alteration by the processing stages in the pipeline.

In accordance with the invention, the interactiv flexibility of the tokens in cooperation with the processing stages facilitates greater functional diversity of the processing stages for resident structure in the pipeline, and the flexibility of the tokens facilitates system expansion and/or alteration. The tokens may be capable of facilitating a plurality of functions within any processing stage in the pipeline. Such pipelin tokens may be either hardware based or software based. Hence, the tokens facilitate more efficient uses of system bandwidth in the pipeline. The tokens may provid data and control simultaneously to the processing stages in the pipelin .

reconfigurable prediction filter which is reconfigurable by a prediction tok n.

Data may be moved along the two-wire interface within the temporal decoder in 8x8 pel data blocks, and address means may be provided for storing and retrieving such data blocks along block boundaries. The address means may store and retrieve blocks of data across block boundaries. The address means reorders said blocks as picture data for display. The data blocks stored and retrieved may be greater and/or smaller than 8x8 pel data blocks. Circuit means may also be provided for either displaying th output of the temporal decoder or writing the output back into a picture memory location. The decoded format may b either a still picture format or a moving picture format.

The above and other objectives and advantages of th invention will become apparent from the following more detailed description.

INTRA CODING: Coding of a macroblock or picture that uses information only from that macroblock or picture.

LUMINANCE (COMPONENT): A matrix, block or single pel representing a monochrome representation of the signal and related to the primary colors in the manner defined in the bit stream. The symbol used for luminance is Y.

MACROBLOCK: The four 8 by 8 blocks of luminance data and the two (for 4:2:0 chroma format) four (for 4:2:2 chroma format) or eight (for 4:4:4 chroma format) corresponding 8 by 8 blocks of chrominance data coming from a 16 by 16 section of the luminance component of the picture. Macroblock is sometimes used to refer to the pel data and sometimes to the coded representation of the pel values and other data elements defined in the macroblock header of the syntax defined in this part of this specification. To one of ordinary skill in the art, the usage is clear from the context.

MOTION COMPENSATION: The use of motion vectors to improve the efficiency of the prediction of pel values. The prediction uses motion vectors to provide offsets into the past and/or future reference pictures containing previously decoded pel values that are used to form the prediction error signal.

MOTION VECTOR: A two-dimensional vector used for motion compensation that provides an offset from the coordinate position in the current picture to the coordinates in a reference picture.

NON-INTRA CODING: Coding of a macroblock or picture that uses information both from itself and from macroblocks and pictures occurring at other times.

PEL: Picture element.

PICTURE: Source, coded or reconstructed image data. A source or reconstructed picture consists of three rectangular matrices of 8-bit numbers representing the luminance and two chrominance signals. For progressive video, a picture is

DETAILED DESCRIPTION OF THE ILLUSTRATIVE EMBODIMENT

As an introduction to the most general features used in a pipeline system which is utilized in the preferred embodiments of the invention, Fig. 1 is a greatly simplified illustration of six cycles of a six-stage pipeline. (As is explained in greater detail below, the preferred embodiment of the pipeline includes several advantageous features not shown in Fig 1.).

Referring now to the drawings, wherein like reference numerals denote like or corresponding elements throughout the various figures of the drawings, and more particularly to Fig. 1, there is shown a block diagram of six cycles in practice of the present invention. Each row of boxes illustrates a cycle and each of the different stages are labelled A-F, respectively. Each shaded box indicates that the corresponding stage holds valid data, i.e., data that is to be processed in one of the pipeline stages. After processing (which may involve nothing more than a simple transfer without manipulation of the data) valid data is transferred out of the pipeline as valid output data.

Note that an actual pipeline application may include more or fewer than six pipeline stages. As will be appreciated, the present invention may be used with any number of pipeline stages. Furthermore, data may be processed in more than one stage and the processing time for different stages can differ.

In addition to clock and data signals (described below), the pipeline includes two transfer control signals -- a "VALID" signal and an "ACCEPT" signal. These signals are used to control the transfer of data within the pipeline. The VALID signal, which is illustrated as the upper of the two lines connecting neighboring stages, is passed in a forward or downstream direction from each pipeline stage to the nearest neighboring device. This device may be another though it does not recognize them. This enables communication between non-adjacent pipeline stages.

Although not shown in Fig. 1, there are data lines, either single lines or several parallel lines, which form a data bus that also lead into and out of each pipeline stage. As is explained and illustrated in greater detail below, data is transferred into, out of, and between the stages of the pipeline over the data lines.

Note that the first pipeline stage may receive data and control signals from any form of preceding device. For example, reception circuitry of a digital image transmission system, another pipeline, or the like. On the other hand, it may generate itself, all or part of the data to be processed in the pipeline. Indeed, as is explained below, a "stage" may contain arbitrary processing circuitry, including none at all (for simple passing of data) or entire systems (for example, another pipeline or even multiple systems or pipelines), and it may generate, change, and delete data as desired.

When a pipeline stage contains valid data that is to be transferred down the pipeline, the VALID signal, which indicates data validity, need not be transferred further than to the immediately subsequent pipeline stage. A two-wire interface is, therefore, included between every pair of pipeline stages in the system. This includes a two-wire interface between a preceding device and the first stage, and between a subsequent device and the last stage, if such other devices are included and data is to be transferred between them and the pipeline.

Each of the signals, ACCEPT and VALID, has a HIGH and a LOW value. These values are abbreviated as "H" and "L", respectively. The most common applications of the pipeline, in practicing the invention, will typically be digital. In such digital implementations, the HIGH value may, for pointing arrow. Hence, the ACCEPT signal from Stage E into Stage D is HIGH, whereas the ACCEPT signal from the device connected downstream of the pipeline into Stage F is LOW.

Data is transferred from one stage to another during a cycle (explained below) whenever the ACCEPT signal of the downstream stage into its upstream neighbor is HIGH. If the ACCEPT signal is LOW between two stages, then data is not transferred between these stages.

Referring again to Fig. 1, if a box is shaded, the corresponding pipeline stage is assumed, by way of example, to contain valid output data. Likewise, the VALID signal which is passed from that stage to the following stage is HIGH. Fig. 1 illustrates the pipeline when stages B, D, and E contain valid data. Stages A, C, and F do not contain valid data. At the beginning, the VALID signal into pipeline stage A is HIGH, meaning that the data on the transmission line into the pipeline is valid.

Also at this time, the ACCEPT signal into pipeline stage F is LOW, so that no data, whether valid or not, is transferred out of Stage F. Note that both valid and invalid data is transferred between pipeline stages. Invalid data, which is data not worth saving, may be written over, thereby, eliminating it from the pipeline. However, valid data must not be written over since it is data that must be saved for processing or use in a downstream device e.g., a pipeline stage, a device or a system connected to the pipeline that receives data from the pipeline.

In the pipeline illustrated in Fig. 1, Stage E contains valid data D1, Stage D contains valid data D2, Stage B contains valid data D3, and a device (not shown) connected to the pipeline upstream contains data D4 that is to be transferred into and processed in the pipeline. Stages B, D and E, in addition to the upstream device, contain valid data and, therefore, the VALID signal from these stages or devices accept new data from Stage F (the ACCEPT signal into Stage F is LOW). Stages E and F, therefore, are still "blocked", but in Cycle 3, Stage D has received the valid data D3, which has overwritten the invalid data that was previously in this stage. Since Stage D cannot pass on data D3 in Cycle 3, it cannot accept new data and, therefore, sets the ACCEPT signal into Stage C LOW. However, stages A-C are ready to accept new data and signal this by setting their corresponding ACCEPT signals HIGH. Note that data D4 has been shifted from Stage A to Stage B.

Assume now that the downstream device becomes ready to accept new data in Cycle 4. It signals this to the pipeline by setting the ACCEPT signal into Stage F HIGH. Although Stages C-F contain valid data, they can now shift the data downstream and are, thus, able to accept new data. Since each stage is therefore able to shift data one step downstream, they set their respective ACCEPT signals out HIGH.

As long as the ACCEPT signal into the final pipeline stage (in this example, Stage F) is HIGH, the pipeline shown in Fig. 1 acts as a rigid pipeline and simply shifts data one step downstream on each cycle. Accordingly, in Cycle 5, data D1, which was contained in Stage F in Cycle 4, is shifted out of the pipeline to the subsequent device, and all other data is shifted one step downstream.

Assume now, that the ACCEPT signal into Stage F goes LOW in Cycle 5. Once again, this means that Stages D-F are not able to accept new data, and the ACCEPT signals out of these stages into their immediately preceding neighbors go LOW. Hence, the data D2, D3 and D4 cannot shift downstream, however, the data D5 can. The corresponding state of the pipeline after Cycle 5 is, thus, shown in Fig. 1 as Cycle 6.

The ability of the pipeline, in accordance with the preferred embodiments of the present invention, to "fill up" illustrated in Fig. 1, for the ACCEPT signal to propagate all the way back to the beginning of the pipeline if there is some intermediate stage that is able to accept new data.

In the embodiment illustrated in Fig. 1, each pipeline stage will still need separate input and output data latches to allow data to be transferred between stages without unintended overwriting. Also, although the pipeline illustrated in Fig. 1 is able to "compress" when downstream pipeline stages are blocked, i.e., they cannot pass on the data they contain, the pipeline does not "expand" to provide stages that contain no valid data between stages that do contain valid data. Rather, the ability to compress depends on there being cycles during which no valid data is presented to the first pipeline stage.

In Cycle 4, for example, if the ACCEPT signal into Stage F remained LOW and valid data filled pipeline stages A and B, as long as valid data continued to be presented to Stage A the pipeline would not be able to compress any further and valid input data could be lost. Nonetheless, the pipeline illustrated in Fig. 1 reduces the risk of data loss since it is able to compress as long as there is a pipeline stage that does not contain valid data.

Fig. 2 illustrates another embodiment of the pipeline that can both compress and expand in a logical manner and which includes circuitry that limits propagation of the ACCEPT signal to the nearest preceding stage. Although the circuitry for implementing this embodiment is explained and illustrated in greater detail below, Fig. 2 serves to illustrate the principle by which it operates.

For ease of comparison only, the input data and ACCEPT signals into the pipeline embodiment shown in Fig. 2 are the same as in the pipeline embodiment shown in Fig. 1. Accordingly, stages E, D and B contain valid data D1, D2 and D3, respectively. The ACCEPT signal into Stage F is LOW; and downstream (the ACCEPT signal into Stage F is LOW) in Cycle 3, Stage E must, therefore, transfer the data D2 into the secondary storage elements of Stage F. Since both the primary and the secondary storage elements of Stage F now contain valid data that cannot be passed on, the ACCEPT signal from Stage F into Stage E is set LOW. Accordingly, this represents a propagation of the LOW ACCEPT signal back only one stage relative to Cycle 2, whereas this ACCEPT signal had to be propagated back all the way to Stage C in the embodiment shown in Fig. 1.

Since Stages A-E are able to pass on their data, the ACCEPT signals from the stages into their immediately preceding neighbors are set HIGH. Consequently, the data D3 and D4 are shifted one stage to the right so that, in Cycle 4, they are loaded into the primary data storage elements of Stage E and Stage C, respectively. Although Stage E now contains valid data D3 in its primary storage elements, its secondary storage elements can still be used to store other data without risk of overwriting any valid data.

Assume now, as before, that the ACCEPT signal into Stage F becomes HIGH in Cycle 4. This indicates that the downstream device to which the pipeline passes data is ready to accept data from the pipeline. Stage F, however, has set its ACCEPT signal LOW and, thus, indicates to Stage E that Stage F is not prepared to accept new data. Observe that the ACCEPT signals for each cycle indicate what will "happen" in the next cycle, that is, whether data will be passed on (ACCEPT HIGH) or whether data must remain in place (ACCEPT LOW). Therefore, from Cycle 4 to Cycle 5, the data D1 is passed from Stage F to the following device, the data D2 is shifted from secondary to primary storage in Stage F, but the data D3 in Stage E is not transferred to Stage F. The data D4 and D5 can be transferred into the following pipeline stages as normal since the following stages have their ACCEPT

LOW. Transfer of data from one storage element to another is indicated by a large open arrow. It is assumed that the VALID signal out of the primary or secondary storage elements of any given stage is HIGH whenever the storage elements contain valid data.

In Fig. 3, each cycle is shown as consisting of a full period of the non-overlapping clock phases ø0 and ø1. As is explained in greater detail below, data is transferred from the secondary storage elements (shown as the left box in each stage) to the primary storage elements (shown as the right box in each stage) during clock cycle ø1, whereas data is transferred from the primary storage elements of one stage to the secondary storage elements of the following stage during the clock cycle ø0. Fig. 3 also illustrates that the primary and secondary storage elements in each stage are further connected via an internal acceptance line to pass an ACCEPT signal in the same manner that the ACCEPT signal is passed from stage to stage. In this way, the secondary storage element will know when it can pass its date to the primary storage element.

Fig. 3 shows the ø1 phase of Cycle 1, in which data D1, D2 and D3, which were previously shifted into the secondary storage elements of Stages E, D and B, respectively, are shifted into the primary storage elements of the respective stage. During the ø1 phase of Cycle 1, the pipeline, therefore, assumes the same configuration as is shown as Cycle 1 of Fig. 2. As before, the ACCEPT signal into Stage F is assumed to be LOW. As Fig. 3 illustrates, however, this means that the ACCEPT signal into the primary storage element of Stage F is LOW, but since this storage element does not contain valid data, it sets the ACCEPT signal into its secondary storage element HIGH.

The ACCEPT signal from the secondary storage elements of Stage F into the primary storage elements of Stage E is also data. It signals this event setting its ACCEPT signal out LOW.

Assuming that the ACCEPT signal into Stage F remains LOW, data upstream of Stage F can continue to be shifted between stages and within stages on the respective clock phases until the next valid data block D3 reaches the primary storage elements of Stage E. As illustrated, this condition is reached during the ø1 phase of Cycle 4.

During the ø0 phase of Cycle 5, data D3 has been loaded into the primary storage element of Stage E. Since this data cannot be shifted further, the ACCEPT signal out of the primary storage elements of Stage E is set LOW. Upstream data can be shifted as normal.

Assume now, as in Cycle 5 of Fig. 2, that the device connected downstream of the pipeline is able to accept pipeline data. It signals this event by setting the ACCEPT signal into pipeline Stage F HIGH during the ø1 phase of Cycle 4. The primary storage elements of Stage F can now shift data to the right and they are also able to accept new data. Hence, the data D1 was shifted out during the ø1 phase of Cycle 5 so that the primary storage elements of Stage F no longer contain data that must be saved. During the ø1 phase of Cycle 5, the data D2 is, therefore, shifted within Stage F from the secondary storage elements to the primary storage elements. The secondary storage elements of Stage F are also able to accept new data and signal this by setting the ACCEPT signal into the primary storage elements of Stage E HIGH. During transfer of data within a stage, that is, from its secondary to its primary storage elements, both sets of storage elements will contain the same data, but the data in the secondary storage elements can be overwritten with no data loss since this data will also be held in the primary storage elements. The same holds true for data transfer from the primary storage elements of one stage into the secondary as illustrated in Fig. 3. The concept of "primary" and "secondary" storage elements is, therefore, mostly a question of labeling. In Fig. 3, the "primary" storage elements can also be referred to as "output" storage elements, since they are the elements from which data is transferred out of a stage into a following stage or device, and the "secondary" storage elements could be "input" storage elements for the same stage.

In explaining the aforementioned embodiments, as shown in Figs. 1-3, only the transfer of data under the control of the ACCEPT and VALID signals has been mentioned. It is to be further understood that each pipeline stage may also process the data it has received arbitrarily before passing it between its internal storage elements or before passing it to the following pipeline stage. Therefore, referring once again to Fig. 3, a pipeline stage can, therefore, be defined as the portion of the pipeline that contains input and output storage elements and that arbitrarily processes data stored in its storage elements.

Furthermore, the "device" downstream from the pipeline Stage F, need not be some other type of hardware structure, but rather it can be another section of the same or part of another pipeline. As illustrated below, a pipeline stage can set its ACCEPT signal LOW not only when all of the downstream storage elements are filled with valid data, but also when a stage requires more than one clock phase to finish processing its data. This also can occur when it creates valid data in one or both of its storage elements. In other words, it is not necessary for a stage simply to pass on the ACCEPT signal based on whether or not the immediately downstream storage elements contains valid data that cannot be passed on. Rather, the ACCEPT signal itself may also be altered within the stage or, by circuitry external to the stage, in order to control the passage of data between adjacent storage change from one stage to the next if a particular application so requires. The interface in accordance with this embodiment can also be used to process analog signals.

As discussed previously, while other conventional timing arrangements may be used, the interface is preferably controlled by a two-phase, non-overlapping clock. In Figs. 4-9, these clock phase signals are referred to as PH0 and PH1. In Fig. 4, a line is shown for each clock phase signal.

Input data enters a pipeline stage over a multi-bit data bus IN_DATA and is transferred to a following pipeline stage or to subsequent receiving circuitry over an output data bus OUT_DATA. The input data is first loaded in a manner described below into a series of input latches (one for each input data signal) collectively referred to as LDIN, which constitute the secondary storage elements described above.

In the illustrated example of this embodiment, it is assumed that the Q outputs of all latches follow their D inputs, that is, they are "loaded", when the clock input is HIGH, i.e., at a logic "1" level. Additionally, the Q outputs hold their last values. In other words, the Q outputs are "latched" on the falling edge of their respective clock signals. Each latch has for its clock either one of two non-overlapping clock signals PH0 or PH1 (as shown in Fig. 5), or the logical AND combination of one of these clock signals PH0, PH1 and one logic signal. The invention works equally well, however, by providing latches that latch on the rising edges of the clock signals, or any other known latching arrangement, as long as conventional methods are applied to ensure proper timing of the latching operations.

The output data from the input data latch LDIN passes via an arbitrary and optional combinatorial logic circuit B1, which may be provided to convert output data from input latch LDIN into intermediate data, which is then later loaded in an output data latch LDOUT, which comprises the primary storage preferably via logic circuitry, which is described below. Similarly, the output QAOUT of the acceptance output latch LAOUT is connected as the input to the acceptance input latch LAIN, preferably via logic circuitry, which is described below.

In practicing the present invention, the output signals QVIN, QVOUT from the validation latches LVIN, LVOUT are combined with the acceptance signals QAOUT, OUT_ACCEPT, respectively, to form the inputs to the acceptance latches LAIN, LAOUT, respectively. In the embodiment illustrated in Fig. 4, these input signals are formed as the logical NAND combination of the respective validation signals QVIN, QVOUT, with the logical inverse of the respective acceptance output signals QAOUT, OUT_ACCEPT. Conventional logic gates, NAND1 and NAND2, perform the NAND operation, and the inverters INV1, INV2 form the logical inverses of the respective acceptance signals.

As is well known in the art of digital design, the output from a NAND gate is a logical "1" when any or all of its input signals are in the logical "0" state. The output from a NAND gate is, therefore, a logical "0" only when all of its inputs are in the logical "1" state. Also well known in the art, is that the output of a digital inverter such as INV1 is a logical "1" when its input signal is a "0" and is a "0" when its input signal is a "1"

The inputs to the NAND gate NAND1 are, therefore, QVIN and NOT (QAOUT), where "NOT" indicates binary inversion. Using known techniques, the input to the acceptance latch LAIN can be resolved as follows:

NAND(QVIN,NOT(QAOUT)) = NOT(QVIN) OR QAOUT

In other words, the combination of the inverter INV1 and the NAND gate NAND1 is a logical "1" either when the signal QVIN is a "0" or the signal QAOUT is a "1", or both. The gate NAND1 and the inverter INV1 can, therefore, be lines of the latches. Consequently, is necessary to provide an actual logic AND gate, which might cause problems of timing due to propagation delay in high-speed applications. The AND gate shown in the figures, therefore, only indicates the logical function to be performed in generating the enable signals of the various latches.

Thus, the data latch LDIN loads input data only when PH0 and QAIN are both "1". It will latch this data when either of these two signals goes to a "0".

Although only one of the clock phase signals PH0 or PH1, is used to clock the data and validation latches at the input (and output) side of the pipeline stage, the other clock phase signal is used, directly, to clock the acceptance latch at the same side. In other words, the acceptance latch on either side (input or output) of a pipeline stage is preferably clocked "out of phase" with the data and validation latches on the same side. For example, PH1 is used to clock the acceptance input latch, although PH0 is used in generating the clock signal CK for the data latch LDIN and the validation latch LVIN.

As an example of the operation of a pipeline augmented by the two-wire validation and acceptance circuitry assume that no valid data is initially presented at the input to the circuit, either from a preceding pipeline stage, or from a transmission device. In other words, assume that the validation input signal IN_VALID to the illustrated stage has not gone to a "1" since the system was most recently reset. Assume further that several clock cycles have taken place since the system was last reset and, accordingly, the circuitry has reached a steady-state condition. The validation input signal QVIN from the validation latch LVIN is, therefore, loaded as a "0" during the next positive period of the clock PH0. The input to the acceptance input latch LAIN (via the gate NAND1 or another equivalent gate, to accept new data, its acceptance signal QAOUT will be a "1". In this case, during the next positive period of the clock signal PH1, the data latch LDOUT and validation latch LVOUT will be enabled, and the data latch LDOUT will load the data present at its input. This will occur before the next rising edge of the other clock signal PH0, since the clock signals are non-overlapping. At the next rising edge of PH0, the preceding data latch (LDIN) will, therefore, not latch in new input data from the preceding stage until the data output latch LDOUT has safely latched the data transferred from the latch LDIN.

Accordingly, the same sequence is followed by every adjacent pair of data latches (within a stage or between adjacent stages) that are able to accept data, since they will be operating based on alternate phases of the clock. Any data latch that is not ready to accept new data because it contains valid data that cannot yet be passed, will have an output acceptance signal (the QA output from its acceptance latch LA) that is LOW, and its data latch LDIN or LDOUT will not be loaded. Hence, as long as the acceptance signal (the output from the acceptance latch) of a given stage or side (input or output) of a stage is LOW, its corresponding data latch will not be loaded.

Fig. 4 also shows a reset feature included in a preferred embodiment. In the illustrated example, a reset signal NOTRESET0 is connected to an inverting reset input R (inversion is hereby indicated by a small circle, as is conventional) of the validation output latch LVOUT. As is well known, this means that the validation latch LVOUT will be forced to output a "0" whenever the reset signal NOTRESET0 becomes a "0". One advantage of resetting the latch when the reset signal goes low (becomes a "0") is that a break in transmission will reset the latches. They will then be in their "null" or reset state whenever a valid transmission necessary to directly reset the validation output latch in the final pipeline stage.

Figs. 5a and 5b (referred to collectively as Fig. 5) illustrate a timing diagram showing the relationship between the non-overlapping clock signals PH0, PH1, the effect of the reset signal, and the holding and transfer of data for the different permutations of validation and acceptance signals into and between the two illustrated sides of a pipeline stage configured in the embodiment shown in Fig. 4. In the example illustrated in the timing diagram of Fig. 5, it has been assumed that the outputs from the data latches LDIN, LDOUT are passed without further manipulation by intervening logic blocks B1, B2. This is by way of example and not necessarily by way of limitation. It is to be understood that any combinatorial logic structures may be included between the data latches of consecutive pipeline stages, or between the input and output sides of a single pipeline stage. The actual illustrated values for the input data (for example the HEX data words "aa" or "04") are also merely illustrative. As is mentioned above, the input data bus may have any width (and may even be analog), as long as the data latches or other storage devices are able to accommodate and latch or store each bit or value of the input word.

Preferred Data Structure - "tokens"

In the sample application shown in Fig. 4, each stage processes all input data, since there is no control circuitry that excludes any stage from allowing input data to pass through its combinatorial logic block B1, B2, and so forth. To provide greater flexibility, the present invention includes a data structure in which "tokens" are used to distribute data and control information throughout the system. Each token consists of a series of binary bits separated into one or more blocks of token words.

Note that no dedicated wires or registers are required to transmit the address field. It is transmitted using the data bits. As is explained below, a pipeline stage will not be slowed down if it is not intended to be activated by the particular address field, i.e., the stage will be able to pass along the token without delay.

The remainder of the data in the token following the address field is not constrained by the use of tokens. These D-data bits may take on any values and the meaning attached to these bits is of no importance here. That is, the meaning of the data can vary, for example, depending upon where the data is positioned within the system at a particular point in time. The number of data bits D appended after the address field can be as long or as short as required, and the number of data words in different tokens may vary greatly. The address field and extension bit are used to convey control signals to the pipeline stages. Because the number of words in the data field (the string of D bits) can be arbitrary, as can be the information conveyed in the data field can also vary accordingly. The explanation below is, therefore, directed to the use of the address and extension bits.

In the present invention, tokens are a particularly useful data structure when a number of blocks of circuitry are connected together in a relatively simple configuration. The simplest configuration is a pipeline of processing steps. For example, in the one shown in Fig. 1. The use of tokens, however, is not restricted to use on a pipeline structure.

Assume once again that each box represents a complete pipeline stage. In the pipeline of Fig. 1, data flows from left to right in the diagram. Data enters the machine and passes into processing Stage A. This may or may not modify the data and it then passes the data to Stage B. The modification, if any, may be arbitrarily complicated and, in general, there will not be the same number of data items set to "0". One alternative to the preferred scheme is to move the extension bit so that it indicates the first word of a token instead of the last. This can be accomplished with appropriate changes in the decoding hardware.

The advantage of using the extension bit of the present invention to signal the last word in a token rather than the first, is that it is often useful to modify the behavior of a block of circuitry depending upon whether or not a token has extension bits. An example of this is a token that activates a stage that processes video quantization values stored in a quantization table (typically a memory device). For example, a table containing 64 eight-bit arbitrary binary integers.

In order to load a new quantization table into the quantizer stage of the pipeline, a "QUANT_TABLE" token is sent to the quantizer. In such a case the token, for example, consists of 65 token words. The first word contains the code "QUANT_TABLE", i.e., build a quantization table. This is followed by 64 words, which are the integers of the quantization table.

When encoding video data, it is occasionally necessary to transmit such a quantization table. In order to accomplish this function, a QUANT_TABLE token with no extension words can be sent to the quantizer stage. On seeing this token, and noting that the extension bit of its first word is LOW, the quantizer stage can read out its quantization table and construct a QUANT_TABLE token which includes the 64 quantization table values. The extension bit of the first word (which was LOW) is changed so that it is HIGH and the token continues, with HIGH extension bits, until the new end of the token, indicated by a LOW extension bit on the sixty fourth quantization table value. This proceeds in the typical way through the system and is encoded into the bit stream.

The advantages of the extension bit scheme in accordance with the present invention include: 1) pipeline stages need not include a block of circuitry that decodes every token since unrecognized tokens can be passed on correctly by considering only the extension bit; 2) the coding of the extension bit is identical for all tokens; 3) there is no limit placed on the length of a token; 4) the scheme is efficient (in terms of overhead to represent the length of the token) for short tokens; and 5) error recovery is naturally achieved. If an extension bit is corrupted then one random token will be generated (for an extension bit corrupted from "1" to "0") or a token will be lost (extension bit corrupted "0" to "1"). Furthermore, the problem is localized to the tokens concerned. After that token, correct operation is resumed automatically.

In addition, the length of the address field may be varied. This is highly advantageous since it allows the most common tokens to be squeezed into the minimum number of words. This, in turn, is of great importance in video data pipeline systems since it ensures that all processing stages can be continuously running at full bandwidth.

In accordance to the present invention, in order to allow variable length address fields, the addresses are chosen so that a short address followed by random data can never be confused with a longer address. The preferred technique for encoding the address field (which also serves as the "code" for activating an intended pipeline stage) is the well-known technique first described by Huffman, hence the common name "Huffman Code". Nevertheless, it will be appreciated by one of ordinary skill in the art, that other coding schemes may also be successfully employed.

Although Huffman encoding is well understood in the field of digital design, the following example provides a general background:

will be able to deal with the new token without modification to their designs because they will not recognize it and will, accordingly, pass that token on unmodified.

This ability of the present invention to leave substantially existing designed devices unaffected has clear advantages. It may be possible to leave some semiconductor chips in a chip set completely unaffected by a design improvement in some other chips in the set. This is advantageous both from the perspective of a customer and from that of a chip manufacturer. Even if modifications mean that all chips are affected by the design change (a situation that becomes increasingly likely as levels of integration progress so that the number of chips in a system drops) there will still be the considerable advantage of better time-to-market than can be achieved, since the same design can be reused.

In particular, note the situation that occurs when it becomes necessary to extend the token set to include two word addresses. Even in this case, it is still not necessary to modify an existing design. Token decoders in the pipeline stages will attempt to decode the first word of such a token and will conclude that it does not recognize the token. It will then pass on the token unmodified using the extension bit to perform this operation correctly. It will not attempt to decode the second word of the token (even though this contains address bits) because it will "assume" that the second word is part of the data field of a token that it does not recognize.

In many cases, a pipeline stage or a connected block of circuitry will modify a token. This usually, but not necessarily, takes the form of modifying the data field of a token. In addition, it is common for the number of data words in the token to be modified, either by removing certain data words or by adding new ones. In some cases, tokens are removed entirely from the token stream.

For the sake of simplicity only, the two-wire interface (with the acceptance and validation signals and latches) is not illustrated and all details dealing with resetting the circuit are omitted. As before, an 8-bit data word is assumed by way of example only and not by way of limitation.

This exemplifying pipeline stage delays the data bits and the extension bit by one pipeline stage. It also decodes the DATA Token. At the point when the first word of the DATA Token is presented at the output of the circuit, the signal "DATA_ADDR" is created and set HIGH. The data bits are delayed by the latches LDIN and LDOUT, each of which is repeated eight times for the eight data bits used in this example (corresponding to an 8-input, 8-output latch). Similarly, the extension bit is delayed by extension bit latches LEIN and LEOUT.

In this example, the latch LEPREV is provided to store the most recent state of the extension bit. The value of the extension bit is loaded into LEIN and is then loaded into LEOUT on the next rising edge of the non-overlapping clock phase signal PH1. Latch LEOUT, thus, contains the value of the current extension bit, but only during the second half of the non-overlapping, two-phase clock. Latch LEPREV, however, loads this extension bit value on the next rising edge of the clock signal PHO, that is, the same signal that enables the extension bit input latch LEIN. The output QEPREV of the latch LEPREV, thus, will hold the value of the extension bit during the previous PHO clock phase.

The five bits of the data word output from the <u>inverting</u> Q output, plus the non-inverted MD[2], of the latch LDIN are combined with the previous extension bit value QEPREV in a series of logic gates NAND1, NAND2, and NOR1, whose operations are well known in the art of digital design. The designation "N_MD[m] indicates the logical inverse of bit m of the mid-data word MD[7:0]. Using known techniques of respectively). In this way, they only hold valid extension bits and are not loaded with spurious values associated with data that is not valid. In the embodiment shown in Fig. 7, the two-wire valid/accept logic includes the OR1 and OR2 gates with input signals consisting of the downstream acceptance signals and the inverting output of the validation latches LVIN and LVOUT, respectively. This illustrates one way in which the gates NAND1/2 and INV1/2 in Fig. 4 can be replaced if the latches have inverting outputs.

Although this is an extremely simple example of a "state-dependent" pipeline stage, i.e., since it depends on the state of only a single bit, it is generally true that all latches holding state information will be updated only when data is actually transferred between pipeline stages. In other words, only when the data is both valid and being accepted by the next stage. Accordingly, care must be taken to ensure that such latches are properly reset.

The generation and use of tokens in accordance with the present invention, thus, provides several advantages over known encoding techniques for data transfer through a pipeline.

First, the tokens, as described above, allow for variable length address fields (and can utilize Huffman coding for example) to provide efficient representation of common tokens.

Second, consistent encoding of the length of a token allows the end of a token (and hence the start of the next token) to be processed correctly (including simple non-manipulative transfer), even if the token is not recognized by the token decoder circuitry in a given pipeline stage.

Third, rules and hardware structures for the handling of unrecognized tokens (that is, for passing them on unmodified) allow communication between one stage and a downstream stage that is not its nearest neighbor in the pipeline. This also interface according to this embodiment can be adapted very easily to different applications.

The duplication stage shown in Fig. 8 also has two latches LEIN and LEOUT that, as in the example shown in Fig. 6, latch the state of the extension bit at the input and at the output of the stage, respectively. As Fig. 8a shows, the input extension latch LEIN is clocked synchronously with the input data latch LDIN and the validation signal IN_VALID.

For ease of reference, the various latches included in the duplication stage are paired below with their respective output signals:

In the duplication stage, the output from the data latch LDIN forms intermediate data referred to as MID_DATA. This intermediate data word is loaded into the data output latch LDOUT only when an intermediate acceptance signal (labeled "MID_ACCEPT" in Fig. 8a) is set HIGH.

The portion of the circuitry shown in Fig. 8 below the acceptance latches LAIN, LAOUT, shows the circuits that are added to the basic pipeline structure to generate the various

S1 may, therefore, only drop to a "0" whenever the last extension bit was "0", indicating that the previous token has ended. Therefore, the MID_DATA word is the first data word in a new token.

The latches LO2 and LI2 together with the NAND gates NAND20 and NAND22 form storage for the signal, DATA_TOKEN. In the normal situation, the signal QI1 at the input to NAND20 and the signal S1 at the input to NAND22 will both be at logic "1". It can be shown, again by the techniques of Boolean algebra, that in this situation these NAND gates operate in the same manner as inverters, that is, the signal QI2 from the output of latch LI2 is inverted in NAND20 and then this signal is inverted again by NAND22 to form the signal S2. In this case, since there are two logical inversions in this path, the signal S2 will have the same value as QI2.

It can also be seen that the signal DATA_TOKEN at the output of latch LO2 forms the input to latch LI2. As a result, as long as the situation remains in which both QI1 and S1 are HIGH, the signal DATA_TOKEN will retain its state (whether "0" or "1"). This is true even though the clock signals PH0 and PH1 are clocking the latches (LI2 and LO2 respectively). The value of DATA_TOKEN can only change when one or both of the signals QI1 and S1 are "0".

As explained earlier, the signal QI1 will be "0" when the previous extension bit was "0". Thus, it will be "0" whenever the MID_DATA value is the first word of a token (and, thus, includes the address field for the token). In this situation, the signal S1 may be either "0" or "1". As explained earlier, signal S1 will be "0" if the MID_DATA word has the predetermined structure that in this example indicates a "DATA" Token. If the MID_DATA word has any other structure, (indicating that the token is some other token, not a DATA Token), S1 will be "1".

HIGH until the DATE_TOKEN signal once again goes to a "1".

The output QO3 (the NOT_DUPLICATE signal) is also fed back and is combined with the output QA1 from the acceptance latch LAIN in a series of logic gates (NAND16 and INV16, which together form an AND gate) that have as their output a "1", only when the signals QA1 and QO3 both have the value "1". As Fig. 8a shows, the output from the AND gate (the gate NAND16 followed by the gate INV16) also forms the acceptance signal, IN_ACCEPT, which is used as described above in the two-wire interface structure.

The acceptance signal IN_ACCEPT is also used as an enabling signal to the latches LDIN, LEIN, and LVIN. As a result, if the NOT_DUPLICATE signal is low, the acceptance signal IN_ACCEPT will also be low, and all three of these latches will be disabled and will hold the values stored at their outputs. The stage will not accept new data until the NOT_DUPLICATE signal becomes HIGH. This is in addition to the requirements described above for forcing the output from the acceptance latch LAIN high.

As long as there is a valid DATA_TOKEN (the DATA_TOKEN signal QO2 is a "1"), the signal QO3 will toggle between the HIGH and LOW states, so that the input latches will be enabled and will be able to accept data, at most, during every other complete cycle of both clock phases PH0, PH1. The additional condition that the following stage be prepared to accept data, as indicated by a "HIGH" OUT_ACCEPT signal, must, of course, still be satisfied. The output latch LDOUT will, therefore, place the same data word onto the output bus OUT_DATA for at least two full clock cycles. The OUT_VALID signal will be a "1" only when there is both a valid DATA_TOKEN (QO2 HIGH) and the validation signal QVOUT is HIGH.

The signal QEIN, which is the extension bit corresponding to MID_DATA, is combined with the signal S3 in a series of to a "1" when the MID_ACCEPT signal is HIGH and when either the validation signal QVIN is high, or when the token is a duplicate (QI3 is a "0"). If the signal MID_ACCEPT is HIGH, the latches LO1-LO3 will, therefore, be enabled when the clock signal PH1 is high whenever valid input data is loaded at the input of the stage, or when the latched data is a duplicate.

From the discussion above, one can see that the stage shown in Figs. 8a and 8b will receive and transfer data between stages under the control of the validation and acceptance signals, as in previous embodiments, with the exception that the output signal from the acceptance latch LAIN at the input side is combined with the toggling duplication signal so that a data word will be output twice before a new word will be accepted.

The various logic gates such as NAND16 and INV16 may, of course, be replaced by equivalent logic circuitry (in this case, a single AND gate). Similarly, if the latches LEIN and LVIN, for example, have inverting outputs, the inverters INV10 and INV12 will not be necessary. Rather, the corresponding input to the gates NAND10 and NAND12 can be tied directly to the inverting outputs of these latches. As long as the proper logical operation is performed, the stage will operate in the same manner. Data words and extension bits will still be duplicated.

One should note that the duplication function that the illustrated stage performs will not be performed unless the first data word of the token has a "1" in the third position of the word and "0's" in the five high-order bits. (Of course, the required pattern can easily be changed and set by selecting other logic gates and interconnections other than the NOR1, NOR2, NND18 gates shown.)

In addition, as Fig. 8 shows, the OUT_VALID signal will be forced low during the entire token unless the first data word aspect of the pres nt invention.

Input latches 34 receive an input over a first bus 31. A first output from the input latches 34 is passed over line 32 to a token decode subsystem 33. A second output from the input latches 34 is passed as a first input over line 35 to a processing unit 36. A first output from the token decode subsystem 33 is passed over line 37 as a second input to the processing unit 36. A second output from the token decode 33 is passed over line 40 to an action identification unit 39. The action identification unit 39 also receives input from registers 43 and 44 over line 46. The registers 43 and 44 hold the state of the machine as a whole. This state is determined by the history of tokens previously received. The output from the action identification unit 39 is passed over line 38 as a third input to the processing unit 36. The output from the processing unit 36 is passed to output latches 41. The output from the output latches 41 is passed over a second bus 42.

Referring now to Figure 11, a Start Code Detector (SCD) 51 receives input over a two-wire interface 52. This input can be either in the form of DATA tokens or as data bits in a data stream. A first output from the Start Code Detector 51 is passed over line 53 to a first logical first-in first-out buffer (FIFO) 54. The output from the first FIFO 54 is logically passed over line 55 as a first input to a Huffman decoder 56. A second output from the Start Code Detector 51 is passed over line 57 as a first input to a DRAM interface 58. The DRAM interface 58 also receives input from a buffer manager 59 over line 60. Signals are transmitted to and received from external DRAM (not shown) by the DRAM interface 58 over line 61. A first output from the DRAM interface 58 is passed over line 62 as a first physical input to the Huffman decoder 56.

data is passed over line 95 to a FIFO 96. The output from the FIFO 96 is then passed over line 97 as a first input to a summer 98. The output from the address generator 94 is passed over line 99 as a first input to a DRAM interface 100. Signals are transmitted to and received from external DRAM (not shown) by the DRAM interface 100 over line 101. A first output from the DRAM interface 100 is passed over line 102 to a prediction filter 103. The output from the prediction filter 103 is passed over line 104 as a second input to the summer 98. A first output from the summer 98 is passed over line 105 to output selector 106. A second output from the summer 98 is passed over line 107 as a second input to the DRAM interface 100. A second output from the DRAM interface 100 is passed over line 108 as a second input to the output selector 106. The output from the output selector 106 is passed over line 109 to a Video Formatter (not shown in Figure 12).

Referring now to Figure 13, a fork 111 receives input from the output selector 106 (shown in Figure 12) over line 112. As a first output from the fork 111, the control tokens are passed over line 113 to an address generator 114. The output from the address generator 114 is passed over line 115 as a first input to a DRAM interface 116. As a second output from the fork 111 the data is passed over line 117 as a second input to the DRAM interface 116. Signals are transmitted to and received from external DRAM (not shown) by the DRAM interface 116 over line 118. The output from the DRAM interface 116 is passed over line 119 to a display pipe 120.

It will be apparent from the above descriptions that each line may comprise a plurality of lines, as necessary.

certain JPEG formats. Additionally, it is shown that the H.261 values and the JPEG values do not overlap, indicating that no single table selection exists that will decode both formats.

Referring now more particularly to Figure 16, there is shown a schematic representation of variable length picture data in accordance with the practice of the present invention. A first picture 161 to be processed contains a first PICTURE_START token 162, first picture information of indeterminate length 163, and a first PICTURE_END token 164. A second picture 165 to be processed contains a second PICTURE_START token 166, second picture information of indeterminate length 167, and a second PICTURE_END token 168. The PICTURE_START tokens 162 and 166 indicate the start of the pictures 161 and 165 to the processor. Likewise, the PICTURE_END tokens 164 and 168 signify the end of the pictures 161 and 165 to the processor. This allows the processor to process picture information 163 and 167 of variable lengths.

Referring to Figure 17, a split 171 receives input over line 172. A first output from the split 171 is passed over line 173 to an address generator 174. The address generated by the address generator 174 is passed over line 175 to a DRAM interface 176. Signals are transmitted to and received from external DRAM (not shown) by the DRAM interface 176 over line 177. A first output from the DRAM interface 176 is passed over line 178 to a prediction filter 179. The output from the prediction filter 179 is passed over line 180 as a first input to a summer 181. A second output from the split 171 is passed over line 182 as an input to a first-in first-out buffer (FIFO) 183. The output from the FIFO 183 is passed over line 184 as a second input to the summer 181. The output from the summer 181 is passed over line 185 to a value register 221 receives image data over a line 222. The line 222 is eight bits wid , allowing for parallel transmission of eight bits at a time. The output from the value register 221 is passed serially over line 223 to a decode register 224. A first output from the decode register 224 is passed to a detector 225 over a line 226. The line 226 is twenty-four bits wide, allowing for parallel transmission of twenty-four bits at a time. The detector 225 detects the presence or absence of an image which corresponds to a standard-independent start code of 23 "zero" values followed by a single "one" value. An 8-bit data value image follows a valid start code image. On detecting the presence of a start code image, the detector 225 transmits a start image over a line 227 to a value decoder 228.

A second output from the decode register 224 is passed serially over line 229 to a value decode shift register 230. The value decode shift register 230 can hold a data value image fifteen bits long. The 8-bit data value following the start code image is shifted to the right of the value decode shift register 230, as indicated by area 231. This process eliminates overlapping start code images, as discussed below. A first output from the value decode shift register 230 is passed to the value decoder 228 over a line 232. The line 232 is fifteen bits wide, allowing for parallel transmission of fifteen bits at a time. The value decoder 228 decodes the value image using a first look-up table (not shown). A second output from the value decode shift register 230 is passed to the value decoder 228 which passes a flag to an index-to-tokens converter 234 over a line 235. The value decoder 228 also passes information to the index-to-tokens converter 234 over a line 236. The information is either the data value image or start code index image obtained from the first look-up table. The flag picture number image is passed over a line 259, an insert image is passed over a line 260, and a replace image is passed over a line 261. The data from the flag generator 251 is passed over a line 262a. A header generator 263 uses a look-up table to generate a replace image, which is passed over a line 262b. An extra word generator 264 uses the MPU to generate an insert image, which is passed over a line 262c. Line 262a, and line 262b combine to form a line 262, which is first input to output latches 265. The output latches 265 pass data over a line 266. The line 266 is fifteen bits wide, allowing for parallel transmission of fifteen bits at a time.

The input valid register (not shown) passes an image as a first input to a first OR gate 267 over a line 268. An insert image is passed over a line 269 as a second input to the first OR gate 267. The output from the first OR gate 267 is passed as a first input to a first AND gate 270 over a line 271. The logical negation of a remove image is passed over a line 272 as a second input to the first AND gate 270 is passed as a second input to the output latches 265 over a line 273. The output latches 265 pass an output valid image over a second two-wire interface 274. An output accept image is received over the second two-wire interface 274 by an output accept latch 275. The output from the output accept latch 275 is passed to an output accept register (not shown) over a line 276.

The output accept register (not shown) passes an image as a first input to a second OR gate 277 over a line 278. The logical negation of the output from the input valid register is passed as a second input to the second OR gate 277 over a line 279. The remove image is passed over a line 280 as a third input to the second OR gate 277. The output from the second OR gate 277 is passed as a first input to a second AND gate 281 over a line 282. The logical negation of

TABLE 600

	Format	Image Received	Tokens Generated
1.	H.261	SEQUENCE START	SEQUENCE START
	MPEG	PICTURE START	GROUP START
	JPEG	(None)	PICTURE START
			PICTURE DATA
2.	H.261	(None)	PICTURE END
	MPEG	(None)	PADDING
	JPEG	(None)	FLUSH
			STOP AFTER PICTURE

As set forth in Table 600 which shows a relationship between the absence or presence of standard signals in the certain machine independent control tokens, the detection of an image by the Start Code Detector 51 generates a sequence of machine independent Control Tokens. Each image listed in the "Image Received" column starts the generation of all machine independent control tokens listed in the group in the "Tokens Generated" column. Therefore, as shown in line 1 of Table 600, whenever a "sequence start" image is received during H.261 processing or a "picture start" image is received during MPEG processing, the entire group of four control tokens is generated, each followed by its corresponding data value or values. In addition, as set forth at line 2 of Table 600, the second group of four control tokens is generated at the proper time irrespective of images received by the Start Code Detector 51.

TABLE 601

DISPLAY ORDER:	I1	B2	B3	P4	B5	B6	P7	B8	B9	I10
TRANSMIT ORDER:	I1	P4	B2	B3	P7	B5	B6	I10	B8	B9

As shown in line 1 of Table 601 which shows the timing relationship between transmitted pictures and displayed pictures, the picture frames are displayed in numerical order. However, in order to reduce the number of frames that

small picture formats and at low coded data rates. The reconfiguration of these DRAMs will be further described hereinafter with reference to the DRAM interface. Typically, a single 4 megabyte DRAM is required by each of the Temporal Decoder and the Spatial Decoder circuits.

The Spatial Decoder of the present invention performs all the required processing within a single picture. This reduces the redundancy within one picture.

The Temporal Decoder reduces the redundancy between the subject picture with relationship to a picture which arrives prior to the arrival of the subject picture, as well as a picture which arrives after the arrival of the subject picture. One aspect of the Temporal Decoder is to provide an address decode network which handles the complex addressing needs to read out the data associated with all of these pictures with the least number of circuits and with high speed and improved accuracy.

As previously described with reference to Figure 11, the data arrives through the Start Code Detector, a FIFO register which precedes a Huffman decoder and parser, through a second FIFO register, an inverse modeller, an inverse quantizer, inverse zigzag and inverse DCT. The two FIFOs need not be on the chip. In one embodiment, the data does not flow through a FIFO that is on the chip. The data is applied to the DRAM interface, and the FIFO-IN storage register and the FIFO-OUT register is off the chip in both cases. These registers, whose operation is entirely independent of the standards, will subsequently be described herein in further detail.

The majority of the subsystems and stages shown in Figure 11 are actually independent of the particular standard used and include the DRAM interface 58, the buffer manager 59 which is generating addresses for the DRAM interface, the inverse modeller 75, the inverse zig-zag 81 and the inverse codes, its operation is essentially independent of the compression standard. For instance, during searching, apart from the circuitry that recognizes the different category of markers, much of the operation is very similar between the three different compression standards.

The next unit is the state machine 68 (Figure 11) located within the Huffman decoder and parser. Here, the actual circuitry is almost identical for each of the three compression standards. In fact, the only element that is affected by the standard in operation is the reset address of the machine. If just the parser is reset, then it jumps to a different address for each standard. There are, in fact, four standards that are recognized. These standards are H.261, JPEG, MPEG and one other, where the parser enters a piece of code that is used for testing. This illustrates that the circuitry is identical in almost every aspect, but the difference is the program in the microcode for each of the standards. Thus, when operating in H.261, one program is running, and when a different program is running, there is no overlap between them. The same holds true for JPEG, which is a third, completely independent program.

The next unit is the Huffman decoder 56 which functions with the index to data unit 64. Those two units cooperate together to perform the Huffman decoding. Here, the algorithm that is used for Huffman decoding is the same, irrespective of the compression standard. The changes are in which tables are used and whether or not the data coming into the Huffman decoder is inverted. Also, the Huffman decoder itself includes a state machine that understands some aspects of the coding standards. These different operations are selected in response to an instruction coming from the parser state machine. The parser state machine operates with a different program for each of the three compression standards

A group of blocks is three rows of macroblocks high by eleven macroblocks wide. In the case of a CIF picture, there are twelve such groups of blocks. However, they are not organized one above the other. Rather, there are two groups of blocks next to each other and then six high, i.e., there are 6 GOB's vertically, and 2 GOB's horizontally.

In all other standards, when performing the addressing, the macroblocks are addressed in order as described above. More specifically, addressing proceeds along the lines and at the end of the line, the next line is started. In H.261, the order of the blocks is the same as described within a group of blocks, but in moving onto the next group of blocks, it is almost a zig-zag.

The present invention provides circuitry to deal with the latter affect. That is the way in which the address generation in the spatial decoder and the video formatter varies for H.261. This is accomplished whenever information is written into the DRAM. It is written with the knowledge of the aforementioned address generation sequence so the place where it is physically located in the RAM is exactly the same as if this had been an MPEG picture of the same size. Hence, all of the address generation circuitry for reading from the DRAM, for instance, when forming predictions, does not have to comprehend that it is H.261 standard because the physical placement of the information in the memory is the same as it would have been if it had been in MPEG sequence. Thus, in all cases, only writing of data is affected.

In the Temporal Decoder, there is an abstraction for H.261 where the circuitry pretends something is different from what is actually occurring. That is, each group of blocks is conceptually stretched out so that instead of having a rectangle which is 11 x 3 macroblocks, the macroblocks are stretched out into a length of 33 blocks (see set, an MPEG set and a JPEG set. Note that there is a much greater overlap between the H.261 set and the MPEG set. They are quite common in the tables they utilize. There is a small overlap between MPEG and JPEG, and there is no overlap at all between H.261 and JPEG so that these standards have totally different sets of tables.

As previously indicated, most of the system units are compression standard independent. If a unit is standard independent, and such units need not remember what CODING_STANDARD is being processed. All of the units that are standard dependent remember the compression standard as the CODING_STANDARD token flows by them. When information encoded/decoded in a first coding standard is distributed through the machine, and a machine is changing standards, prior machines under microprocessor control would normally choose to perform in accordance with the H.261 compression standard. The MPU in such prior machines generates signals stating in multiple different places within the machine that the compression standard is changing. The MPU makes changes at different times and, in addition, may flush the pipeline through.

In accordance with the invention, by issuing a change of CODING_STANDARD tokens at the Start Code Detector that is positioned as the first unit in the pipeline, this change of compression standard is readily handled. The token says a certain coding standard is beginning and that control information flows down the machine and configures all the other registers at the appropriate time. The MPU need not program each register.

The prediction token signals how to form predictions using the bits in the bitstream. Depending on which compression standard is operating, the circuitry translates the information that is found in the standard, i.e. from the bitstream into a prediction mode token. This processing is encoded video signals through the use of techniques all compatible with the single pipeline decoder and processing system. The Spatial Decoder is combined with the Temporal Decoder, and the Video Formatter is used in driving a video display.

Another aspect of the invention is the use of the combination of the Spatial Decoder and the Video Formatter for use with only still pictures. The compression standard independent Spatial Decoder performs all of the data processing within the boundaries of a single picture. Such a decoder handles the spatial decompression of the internal picture data which is passing through the pipeline and is distributed within associated random access memories, standard independent address generation circuits for handling the storage and retrieval of information into the memories. Still picture data is decoded at the output of the Spatial Decoder, and this output is employed as input to the multi-standard, configurable Video Formatter, which then provides an output to the display terminal. In a first sequence of similar pictures, each decompressed picture at the output of the Spatial Decoder is of the same length in bits by the time the picture reaches the output of the Spatial Decoder. A second sequence of pictures may have a totally different picture size and, hence, have a different length when compared to the first length. Again, all such second sequence of similar pictures are of the same length in bits by the time such pictures reach the output of the Spatial Decoder.

Another aspect of the invention is to internally organize the incoming standard dependent bitstream into a sequence of control tokens and DATA tokens, in combination with a plurality of sequentially-positioned reconfigurable processing stages selected and organized to act as a standard-independent, reconfigurable-pipeline-processor.

identified components, Start Code Detector, memory buffers, and Huffman decoder enhances the handling of certain sequences in the input bitstream.

In addition, off chip DRAMs are used for decoding JPEG-encoded video pictures in real time. The size and speed of the buffers used with the DRAMs will depend on the video encoded data rates.

The coding standards identify all of the standard dependent types of information that is necessary for storage in the DRAMs associated with the Spatial Decoder using standard independent circuitry.

3. MOTION PICTURE DECOMPRESSION

In the present invention, if motion pictures are being decompressed through the steps of decoding, a further Temporal Decoder is necessary. The Temporal Decoder combines the data decoded in the Spatial Decoder with pictures, previously decoded, that are intended for display either before or after the picture being currently decoded. The Temporal Decoder receives, in the picture coded datastream, information to identify this temporally-displaced information. The Temporal Decoder is organized to address temporally and spatially displaced information, retrieve it, and combine it in such a way as to decode the information located in one picture with the picture currently being decoded and ending with a resultant picture that is complete and is suitable for transmission to the video formatter for driving the display screen. Alternatively, the resultant picture can be stored for subsequent use in temporal decoding of subsequent pictures.

Generally, the Temporal Decoder performs the processing between pictures either earlier and/or later in time with reference to the picture currently being decoded. The Temporal Decoder reintroduces information that is not encoded within the coded representation of the picture, because it is

The Temporal Decoder also reorders the blocks of picture data for display by a display circuit. The address decode circuitry, described hereinafter, provides handling of this reordering.

As previously mentioned, one important feature of the Temporal Decoder is to add picture information together from a selection of pictures which have arrived earlier or later than the picture under processing. When a picture is described in this context, it may mean any one of the following:

1. The coded data representation of the picture;
2. The result, i.e., the final decoded picture resulting from the addition of a process step performed by the decoder;
3. Previously decoded pictures read from the DRAM; and
4. The result of the spatial decoding, i.e., the extent of data between a PICTURE_START token and a subsequent PICTURE_END token.

After the picture data information is processed by the Temporal Decoder, it is either displayed or written back into a picture memory location. This information is then kept for further reference to be used in processing another different coded data picture.

Re-ordering of the MPEG encoded pictures for visual display involves the possibility that a desired scrambled picture can be achieved by varying the re-ordering feature of the Temporal Decoder.

4. RAM MEMORY MAP

The Spatial Decoder, Temporal Decoder and Video Formatter all use external DRAM. Preferably, the same DRAM is used for all three devices. While all three devices use DRAM, and all three devices use a DRAM interface in conjunction with an address generator, what each implements picture buffers for later use in decoding P and B pictures. Decoding P pictures requires forming predictions from a previously decoded P or I picture. The decoded P picture is stored in a picture buffer for use decoding P and B pictures. B pictures can require predictions form both of the picture buffers. However, B pictures are not stored in the external DRAM.

Note that I and P pictures are not output from the Temporal Decoder as they are decoded. Instead, I and P pictures are written into one of the picture buffers, and are read out only when a subsequent I or P picture arrives for decoding. In other words, the Temporal Decoder relies on subsequent P or I pictures to flush previous pictures out of the two picture buffers, as further discussed hereinafter in the section on flushing. In brief, the Spatial Decoder can provide a fake I or P picture at the end of a video sequence to flush out the last P or I picture. In turn, this fake picture is flushed when a subsequent video sequence starts.

The peak memory band width load occurs when decoding B pictures. The worst case is the B frame may be formed from predictions from both the picture buffers, with all predictions being made to half-pixel accuracy.

As previously described, the Temporal Decoder can be configured to provide MPEG picture reordering. With this picture reordering, the output of P and I pictures is delayed until the next P or I picture in the data stream starts to be decoded by the Temporal Decoder.

As the P or I pictures are reordered, certain tokens are stored temporarily on chip as the picture is written into the picture buffers. When the picture is read out for display, these stored tokens are retrieved. At the output of the Temporal Decoder, the DATA Tokens of the newly decoded P or I picture are replaced with DATA Tokens for the older P or I picture.

PICTURE_END token, but still of widely varying l ngth. There may be other information transmitted here (between the first and second picture), but it is known that the first pictur has finished.

The data stream at the output of the Spatial Decoder consists of pictures, still with picture-starts and picture-ends, of the same length (number of bits) for a given sequence. The length of time between a picture-start and a picture-end may vary.

The Video Formatter takes these pictures of non-uniform time and displays them on a screen at a fixed picture rate determined by the type of display being driven. Different display rates are used throughout the world, e.g. PAL-NTSC television standards. This is accomplished by selectively dropping or repeating pictures in a manner which is unique. Ordinary "frame rate converters," e.g. 2-3 pulldown, operate with a fixed input picture rate, whereas the Video Formatter can handle a variable input picture rate.

6. RECONFIGURABLE PROCESSING STAGE

Referring again to Figure 10, the reconfigurable processing stage (RPS) comprises a token decode circuit 33 which is employed to receive the tokens coming from a two wire interface 37 and input latches 34. The output of the token decode circuit 33 is applied to a processing unit 36 over the two-wire interface 37 and an action identification circuit 39. The processing unit 36 is suitable for processing data under the control of the action identification circuit 39. After the processing is completed, the processing unit 36 connects such completed signals to the output, two-wire interface bus 40 through output latches 41.

The action identification decode circuit 39 has an input from the token decode circuit 33 over the two-wire

Control tokens may also be processed.

A more detailed description of the various types of tokens usable in the present invention will be subsequently described hereinafter. For the purpose of this portion of the specification, it is sufficient to note that the address carried by the control token is decoded in the decoder 33 and is used to access registers contained within the action identification circuit 39. When the token being examined is a recognized control token, the action identification circuit 39 uses its reconfiguration state circuit for distributing the control signals throughout the state machine. As previously mentioned, this activates the state machine of the action identification decoder 39, which then reconfigures itself. For example, it may change coding standards. In this way, the action identification circuit 39 decodes the required action for handling the particular standard now passing through the state machine shown with reference to Figure 10.

Similarly, the processing unit 36 which is under the control of the action identification circuit 39 is now ready to process the information contained in the data fields of the DATA token when it is appropriate for this to occur. On many occasions, a control token arrives first, reconfigures the action identification circuit 39 and is immediately followed by a DATA token which is then processed by the processing unit 36. The control token exits the output latches circuit 41 over the output two-wire interface 42 immediately preceding the DATA token which has been processed within the processing unit 36.

In the present invention, the action identification circuit, 39, is a state machine holding history state. The registers, 43 and 44 hold information that has been decoded from the token decoder 33 and stored in these registers.

might ~~be~~ viewed by one RPS unit as DATA Tokens while another RPS unit might decide that it is actually a Control Token. For example, the quantization table information, as far as the Huffman decoder and state machine is concerned, is data, because it arrives on its input as coded data, it gets formatted up into a series of 8 bit words, and they get formed into a token called a quantization table token (QUANT_TABLE) which goes down the processing pipeline. As far as that machine is concerned, all of that was data; it was handling data, transforming one sort of data into another sort of data, which is clearly a function of the processing performed by that portion of the machine. However, when that information gets to the inverse quantizer, it stores the information in that token a plurality of registers. In fact, because there are 64 8-bit numbers and there are many registers, in general, many registers may be present. This information is viewed as control information, and then that control information affects the processing that is done on subsequent DATA tokens because it affects the number that you multiply each data word. There is an example where one stage viewed that token as being data and another stage viewed it as being control.

Token data, in accordance with the invention is almost universally viewed as being data through the machine. One of the important aspects is that, in general, each stage of circuitry that has a token decoder will be looking for a certain set of tokens, and any tokens that it does not recognize will be passed unaltered through the stage and down the pipeline, so that subsequent stages downstream of the current stage have the benefit of seeing those tokens and may respond to them. This is an important feature, namely there can be communication between blocks that are not adjacent to one another using the token mechanism.

Another important feature of the invention is that each of a constant, K. Another flag tells the inverse quantizer whether to add another constant. The inverse quantizer remembers in a register the CODING_STANDARD token as it flows by the quantizer. When DATA tokens pass thereafter, the inverse quantizer remembers what the standard is and it looks up the parameters that it needs to apply to the processing elements in order to perform a proper operation. For example, the inverse quantizer will look up whether K is set to 0, or whether it is set to 1 for a particular compression standard, and will apply that to its processing circuitry.

In a similar sense the Huffman decoder 56 has a number of tables within it, some for JPEG, some for MPEG and some for H.261. The majority of those tables, in fact, will service more than one of those compression standards. Which tables are used depends on the syntax of the standard. The Huffman decoder works by receiving a command from the state machine which tells it which of the tables to use. Accordingly, the Huffman decoder does not itself directly have a piece of state going into it, which is remembered and which says what coding it is performing. Rather, it is the combination of the parser state machine and Huffman decoder together that contain information within them.

Regarding the Spatial Decoder of the present invention, the address generation is modified and is similar to that shown in Figure 10, in that a number of pieces of information are decoded from tokens, such as the coding standard. The coding standard and additional information as well, is recorded in the registers and that affects the progress of the address generator state machine as it steps through and counts the macroblocks in the system, one after the other. The last stage would be the prediction filter 179 (Figure 17) which operates in one of two modes, either H.261 or MPEG and are easily identified.

form of the invention, the token extension is used to carry the current coding standard which is decoded by the relative token decode circuits distributed throughout the machine, and is used to reconfigure the action identification circuit 39 of stages throughout the machine wherever it is appropriate to operate under a new coding standard. Additionally, the token decode circuit can indicate whether a control token is related to one of the selected standards which the circuit was designed to handle.

More specifically, an MPEG start code and a JPEG marker are followed by an 8 bit value. The H.261 start code is followed by a 4 bit value. In this context, the Start Code Detector 51, by detecting either an MPEG start-code or a JPEG marker, indicates that the following 8 bits contain the value associated with the start-code. Independently, it can then create a signal which indicates that it is either an MPEG start code or a JPEG marker and not an H.261 start code. In this first instance, the 8 bit value is entered into a decode circuit, part of which creates a signal indicating the index and flag which is used within the current circuit for handling the tokens passing through the circuit. This is also used to insert portions of the control token which will be looked at thereafter to determine which standard is being handled. In this sense, the control token contains a portion indicating that it is related to an MPEG standard, as well as a portion which indicates what type of operation should be performed on the accompanying data. As previously discussed, this information is utilized in the system to reconfigure the processing stage used to perform the function required by the various standards created for that purpose.

For example, with reference to the H.261 start code, it is associated with a 4 bit value which follows immediately after the start code. The Start Code Detector passes this value into the token generator state machine. The value is through the decoder to the display.

8. MULTI-STANDARD PROCESSING CIRCUIT - SECOND MODE OF OPERATION

A compression standard-dependent circuit, in the form of the previously described Start Code Detector, is suitably interconnected to a compression standard-independent circuit over an appropriate bus. The standard-dependent circuit is connected to a combination dependent-independent circuit over the same bus and an additional bus. The standard-independ nt circuit applies additional input to the standard dependent-independent circuit, while the latter provides information back to the standard-independent circuit. Information from the standard-independent circuit is applied to the output over another suitable bus. Table 600 illustrates that the multiple standards applied as the input to the standard-dependent Start Code Detector 51 include certain bit streams which have standard-dependent meanings within each encoded bit stream.

9. START-CODE DETECTOR

As previously indicated the Start Code Detector, in accordance with the present invention, is capable of taking MPEG, JPEG and H.261 bit streams and generating from them a sequence of proprietary tokens which are meaningful to the rest of the decoder. As an example of how multi-standard decoding is achieved, the MPEG (1 and 2) picture_start_code, the H.261 picture_start_code and the JPEG start_of_scan (SOS) marker are treated as equivalent by the Start Code Detector, and all will generate an internal PICTURE_START token. In a similar way, the MPEG sequence_start_code and the JPEG SOI (start_of_image) marker both generate a machine sequence_start_token. The H.261 standard, however, has no equivalent start code. Accordingly, the Start Code Detector, by contents of the tokens, and are prepared to handle data which is expected to be received when the picture DATA Token arrives at that station.

As previously described, one of the compression standards, such as H.261, does not have a sequence_start image in its data stream, nor does it have a PICTURE_END image in its data stream. The Start Code Detector indicates the PICTURE_END point in the incoming bit stream and creates a PICTURE_END token. In this regard, the system of the present invention is intended to carry data words that are fully packed to contain a bit of information in each of the register positions selected for use in the practice of the present invention. To this end, 15 bits have been selected as the number of bits which are passed between two start codes. Of course, it will be appreciated by one of ordinary skill in the art, that a selection can be made to include either greater or fewer than 15 bits. In other words, all 15 bits of a data word being passed from the Start Code Detector into the DRAM interface are required for proper operation. Accordingly, the Start Code Detector creates extra bits, called padding, which it inserts into the last word of a DATA Token. For purposes of illustration 15 data bits has been selected.

To perform the Padding operation, in accordance with the present invention, binary 0 followed by a number of binary 1's are automatically inserted to complete the 15 bit data word. This data is then passed through the coded data buffer and presented to the Huffman decoder, which removes the padding. Thus, an arbitrary number of bits can be passed through a buffer of fixed size and width.

In one embodiment, a slice_start control token is used to identify a slice of the picture. A slice_start control token is employed to segment the picture into smaller regions. The size of the region is chosen by the encoder, synta~~cti~~c differences between the coding standards and to function in co-operation with the error conditions. The automatic token generation is done after the serial analysis of the standard-dependent data. Therefore, the Spatial Decoder responds equally to tokens that have been supplied directly to the input of the Spatial Decoder, i.e. the SCD, as well as to tokens that have been generated following the detection of the start-codes in the coded data. A sequence of extra tokens is inserted into the two- wire interface in order to control the multi-standard nature of the present invention.

The MPEG and H.261 coded video streams contain standard dependent, non-data, identifiable bit patterns, one of which is hereinafter called a start image and/or standard-dependent code. A similar function is served in JPEG, by marker codes. These start/marker codes identify significant parts of the syntax of the coded datastream. The analysis of start/marker codes performed by the Start Code Detector is the first stage in parsing the coded data.

The start/marker code patterns are designed so that they can be identified without decoding the entire bit stream. Thus, they can be used, in accordance with the present invention, to assist with error recovery and decoder start-up. The Start Code Detector provides facilities to detect errors in the coded data construction and to assist the start-up of the decoder. The error detection capability of the Start Code Detector will subsequently be discussed in further detail, as will the process of starting up of the decoder.

The aforementioned description has been concerned primarilty with the characteristics of the machine-dependent bit stream and its relationship with the addressing characteristics of the present invention. The following description is of the bit stream characteristics of the where compatibility is required for multiple standards, the system has been optimized for handling all functions in all standards. Accordingly, in many situations, unique start control tokens must be created which are compatible not only with the values contained in the values of the encoded signal standard image, but which are also capable of controlling the various stages to emulate the operation of the standard as represented by specified parameters for each standard which are well known in the art. All such standards are incorporated by reference into this specification.

It is important to understand the relationship between tokens which, alone or in combination with other control tokens, emulate the nondata information contained in the standard bit stream. A separate set of index signals, including flag signals, are generated by each state machine to handle some of the processing within that state machine. Values carried in the standards can be used to access machine dependent control signals to emulate the handling of the standard data and non-data signals. For example, the slice_start token is a two word token, and it is then entered onto the two wire interface as previously described.

The data input to the system of the present invention may be a data source from any suitable data source such as disk, tape, etc., the data source providing 8 bit data to the first functional stage in the Spatial Decoder, the Start Code Detector 51 (Figure 11). The Start Code Detector includes three shift registers; the first shift register is 8 bits wide, the next is 24 bits wide, and the next is 15 bits wide. Each of the registers is part of the two-wire interface. The data from the data source is loaded into the first register as a single 8 bit byte during one timing cycle. Thereafter, the contents of the first shift register is shifted one bit at a time into the decode (second) shift register. After 24 cycles, the 24 bit register is full.

processing stage (RPS) which is a stage, which in response to a recognized token, reconfigures itself to perform various operations.

Tokens may be either position dependent or position independent upon the processing stages for performance of various functions. Tokens may also be metamorphic in that they can be altered by a processing stage and then passed down the pipeline for performance of further functions. Tokens may interact with all or less than all of the stages and in this regard may interact with adjacent and/or non-adjacent stages. Tokens may be position dependent for some functions and position independent for other functions, and the specific interaction with a stage may be conditioned by the previous processing history of a stage.

A PICTURE_END token is a way of signalling the end of a picture in a multi-standard decoder.

A multi-standard token is a way of mapping MPEG, JPEG and H.261 data streams onto a single decoder using a mixture of standard dependent and standard independent hardware and control tokens.

A SEARCH_MODE token is a technique for searching MPEG, JPEG and H.261 data streams which allows random access and enhanced error recovery.

A STOP_AFTER_PICTURE token is a method of achieving a clear end to decoding which signals the end of a picture and clears the decoder pipeline, i.e., channel change.

Furthermore, padding a token is a way of passing an arbitrary number of bits through a fixed size, fixed width buffer.

The present invention is directed to a pipeline processing system which has a variable configuration which uses tokens and a two-wire system. The use of control tokens and DATA Tokens in combination with a two-wire system facilitates a multi-standard system capable of having one or more words. The width of the token is changeable and can be selected as any number of bits. An extension bit indicates whether a token is extended beyond the current word, i.e., if it is set to binary one in all words of a token, except the last word of a token. If the first word of a token has an extension bit of zero, this indicates that the token is only one word long.

Each token is identified by an address field that starts at bit 7 of the first word of the token. The address field is variable in length and can potentially extend over multiple words. In a preferred embodiment, the address is no longer than 8 bits long. However, this is not a limitation on the invention, but on the magnitude of the processing steps elected to be accomplished by use of these tokens. It is to be noted under the extension bit identification label that the extension bit in words 1 and 2 is a 1, signifying that additional words will be coming thereafter. The extension bit in word 3 is a zero, therefore indicating the end of that token.

The token is also capable of variable bit length. For example, there are 9 bits in the token word plus the extension bit for a total of 10 bits. In the design of the present invention, output buses are of variable width. The output from the Spatial Decoder is 9 bits wide, or 10 bits wide when the extension bit is included. In a preferred embodiment, the only token that takes advantage of these extra bits is the DATA token; all other tokens ignore this extra bit. It should be understood that this is not a limitation, but only an implementation.

Through the use of the DATA token and control token configuration, it is possible to vary the length of the data being carried by these DATA tokens in the sense of the number of bits in one word. For example, it has been discussed that data bits in word of a DATA Token can be combined with the this variable picture rate to a constant picture rate suitable for display. However, the picture data is still carried by DATA tokens consisting of 64 words.

11. DRAM INTERFACE

A single high performance, configurable DRAM interface is used on each of the 3 decoder chips. In general, the DRAM interface on each chip is substantially the same; however, the interfaces differ from one to another in how they handle channel priorities. This interface is designed to directly drive the external DRAMs used by the Spatial Decoder, the Temporal Decoder and the Video Formatter. Typically, no external logic, buffers or components will be required to connect the DRAM interface to the DRAMs in those systems.

In accordance with the present invention, the interface is configurable in two ways:

1. The detailed timing of the interface can be configured to accommodate a variety of different DRAM types.
2. The width of the data interface to the DRAM can be configured to provide a cost/performance trade off for different applications.

In general, the DRAM interface is a standard-independent block implemented on each of the three chips in the system. Again, these are the Spatial Decoder, Temporal Decoder and video formatter. Referring again to Figures 11, 12 and 13, these figures show block diagrams that depict the relationship between the DRAM interface, and the remaining blocks of the Spatial Decoder, Temporal Decoder and video formatter, respectively. On each chip, the DRAM interface connects the chip to an external DRAM. External DRAM is used because, at present, it is not practical to fabricate on chip the relatively large amount of DRAM needed. Note: each chip has its own external DRAM and its own DRAM interface.

swing buffer.

In the present invention, each of the chips has four swing buffers, but the function of these swing buffers is different in each case. In the spatial decoder, one swing buffer is used to transfer coded data to the DRAM, another to read coded data from the DRAM, the third to transfer tokenized data to the DRAM and the fourth to read tokenized data from the DRAM. In the Temporal Decoder, however, one swing buffer is used to write intra or predicted picture data to the DRAM, the second to read intra or predicted data from the DRAM and the other two are used to read forward and backward prediction data. In the video formatter, one swing buffer is used to transfer data to the DRAM and the other three are used to read data from the DRAM, one for each of luminance (Y) and the red and blue color difference data (Cr and Cb, respectively).

The following section describes the operation of a hypothetical DRAM interface which has one write swing buffer and one read swing buffer. Essentially, this is the same as the operation of the Spatial Decoder's DRAM interface. The operation is illustrated in Figure 23.

Figure 23 illustrates that the control interfaces between the address generator 301, the DRAM interface 302, and the remaining stages of the chip which pass data are all two wire interfaces. The address generator 301 may either generate addresses as the result of receiving control tokens, or it may merely generate a fixed sequence of addresses (e.g., for the FIFO buffers of the Spatial Decoder). The DRAM interface treats the two wire interfaces associated with the address generator 301 in a special way. Instead of keeping the accept line high when it is ready to receive an address, it waits for the address generator to supply a valid address, processes that address and then sets the accept line high for one clock period. Thus, it implements a diagram of a write swing buffer. The write swing buffer interface includ s two blocks of RAM, RAM1 311 and RAM2 312. As discussed further herein, data is written into RAM1 311 and RAM2 312 from the previous stage, under the control of the write address 313 and control 314. From RAM1 311 and RAM2 312, the data is written into DRAM 515. When writing data into DRAM 315, the DRAM row address is provided by the address generator, and the column address is provided by the write address and control, as described further herein. In operation, valid data is presented at the input 316 (data in). Typically, the data is received from the previous stage. As each piece of data is accepted by the DRAM interface, it is written into RAM1 311 and the write address control increments the RAM1 address to allow the next piece of data to be written into RAM1. Data continues to be written into RAM1 311 until either there is no more data, or RAM1 is full. When RAM1 311 is full, the input side gives up control and sends a signal to the read side to indicate that RAM1 is now ready to be read. This signal passes between two asynchronous clock regimes and, therefore, passes through three synchronizing flip flops.

Provided RAM2 312 is empty, the next item of data to arrive on the input side is written into RAM2. Otherwise, this occurs when RAM2 312 has emptied. When the round robin or priority encoder (depending on which is used by the particular chip) indicates that it is now the turn of this swing buffer to be read, the DRAM interface reads the contents of RAM1 311 and writes them to the external DRAM 315. A signal is then sent back across the asynchronous interface, to indicate that RAM1 311 is now ready to be filled again.

If the DRAM interface empties RAM1 311 and "swings" it before the input side has filled RAM2 312 , then data can be

Temporal Decoder and the Video Formatter. The Temporal Decoder's addressing is more complex because of its predictive aspects as discussed further in this section. The video formatter's addressing is more complex because of multiple video output standard aspects, as discussed further in the sections relating to the video formatter.

As mentioned previously, the Temporal Decoder has four swing buffers: two are used to read and write decoded intra and predicted (I and P) picture data. These operate as described above. The other two are used to receive prediction data. These buffers are more interesting.

In general, prediction data will be offset from the position of the block being processed as specified in the motion vectors in x and y. Thus, the block of data to be retrieved will not generally correspond to the block boundaries of the data as it was encoded (and written into the DRAM). This is illustrated in Figure 25, where the shaded area represents the block that is being formed whereas the dotted outline represents the block from which it is being predicted. The address generator converts the address specified by the motion vectors to a block offset (a whole number of blocks), as shown by the big arrow, and a pixel offset, as shown by the little arrow.

In the address generator, the frame pointer, base block address and vector offset are added to form the address of the block to be retrieved from the DRAM. If the pixel offset is zero, only one request is generated. If there is an offset in either the x or y dimension then two requests are generated, i.e., the original block address and the one immediately below. With an offset in both x and y, four requests are generated. For each block which is to be retrieved, the address generator calculates start and stop addresses which is best illustrated by an example.

Consider a pixel offset of (1,1), as illustrated by the buffer address register is loaded with the inverse of the stop value. The y inverse stop value forms the 3 MSBs and the x inverse stop value forms the 3 LSB. In this case, while the DRAM interface is reading address 9 in the external DRAM, the swing buffer address is zero. The swing buffer address register is then incremented as the external DRAM address register is incremented, as consistent with proper prediction addressing.

The discussion so far has centered on an 8 bit DRAM interface. In the case of a 16 or 32 bit interface, a few minor modifications must be made. First, the pixel offset vector must be "clipped" so that it points to a 16 or 32 bit boundary. In the example we have been using, for block A, the first DRAM read will point to address 0, and data in addresses 0 through 3 will be read. Second, the unwanted data must be discarded. This is performed by writing all the data into the swing buffer (which must now be physically larger than was necessary in the 8 bit case) and reading with an offset. When performing MPEG half-pel interpolation, 9 bytes in x and/or y must be read from the DRAM interface. In this case, the address generator provides the appropriate start and stop addresses. Some additional logic in the DRAM interface is used, but there is no fundamental change in the way the DRAM interface operates.

The final point to note about the Temporal Decoder DRAM interface of the present invention, is that additional information must be provided to the prediction filters to indicate what processing is required on the data. This consists of the following:

a "last byte" signal indicating the last byte of a transfer (of 64,72 or 81 bytes);

an H.261 flag;

a bidirectional prediction flag;

two bits to indicate the block's dimensions (8 or 9 bytes above), and the stop value is compared with the start value to generate the signal which indicates when reading should stop.

The DRAM interface timing block in the present invention uses timing chains to place the edges of the DRAM signals to a precision of a quarter of the system clock period. Two quadrature clocks from the phase locked loop are used. These are combined to form a notional 2x clock. Any one chain is then made from two shift registers in parallel, on opposite phases of the 2x clock.

First of all, there is one chain for the page start cycle and another for the read/write/refresh cycles. The length of each cycle is programmable via the microprocessor interface, after which the page start chain has a fixed length, and the cycle chain's length changes as appropriate during a page start.

On reset, the chains are cleared and a pulse is created. The pulse travels along the chains and is directed by the state information from the DRAM interface. The pulse generates the DRAM interface clock. Each DRAM interface clock period corresponds to one cycle of the DRAM, consequently, as the DRAM cycles have different lengths, the DRAM interface clock is not at a constant rate.

Moreover, additional timing chains combine the pulse from the above chains with the information from the DRAM interface to generate the output strobes and enables such as notcas, notras, notwe, notbe.

12. PREDICTION FILTERS

Referring again to Figures 12, 17, 18, and more particularly to Figure 12, there is shown a block diagram of the Temporal Decoder. This includes the prediction filter. The relationship between the prediction filter and the rest of the elements of the temporal decoder is shown in greater to perform no filtering at all in JPEG mode. As with many other reconfigurable aspects of the three chip system, the prediction filter is reconfigured by means of tokens. Tokens are also used to inform the address generator of the particular mode of operation. In this way, the address generator can supply the prediction filter with the addresses of the needed data, which varies significantly between MPEG and JPEG.

13. ACCESSING REGISTERS

Most registers in the microprocessor interface (MPI) can only be modified if the stage with which they are associated is stopped. Accordingly, groups of registers will typically be associated with an access register. The value zero in an access register indicates that the group of registers associated with that particular access register should not be modified. Writing 1 to an access register requests that a stage be stopped. The stage may not stop immediately, however, so the stages access register will hold the value, zero, until it is stopped.

Any user software associated with the MPI and used to perform functions by way of the MPI should wait "after writing a 1 to a request access register" until 1 is read from the access register. If a user writes a value to a configuration register while its access register is set to zero, the results are undefined.

14. MICRO-PROCESSOR INTERFACE

A standard byte wide micro-processor interface (MPI) is used on all circuits with in the Spatial Decoder and Temporal Decoder. The MPI operates asynchronously with various Spatial and Temporal Decoder clocks. Referring to Table A.6.1 of the subsequent further detailed description, there is shown the various MPI signals that are shown with reference to Figure 54. This Figure shows each individual signal name as associated with the MPI write timing. The name, the characteristic of the signal, and other various physical characteristics are shown with reference to Table 6.6.

17. KEYHOLE ADDRESS LOCATIONS

In the present invention, certain less frequently accessed memory map locations have been placed behind keyhole registers. A keyhole register has two registers associated with it. The first register is a keyhole address register and the second register is a keyhole data register. The keyhole address specifies a location within a extended address space. A read or a write operation to a keyhole data register accesses the locations specified by the keyhole address register. After accessing a keyhole data register, the associated keyhole address register increments. Random access within the extended address space is only possible by writing in a new value to the keyhole address register for each access. A circuit within the present invention may have more than one keyhole memory maps. Nonetheless, there is no interaction between the different keyholes.

18. PICTURE-END

Referring again to Figure 11, there is shown a general block diagram of the Spatial Decoder used in the present invention. It is through the use of this block diagram that the function of PICTURE_END will be described. The PICTURE_END function has the multi-standard advantage of being able to handle H.261 encoded picture information, MPEG and JPEG signals.

As previously described, the system of Figure 11 is interconnected by the two wire interface previously demultiplexor, the end of picture even though it has not had the typically expected full range and/or number of signals applied to the Huffman decoder and video demultiplexor circuit. In this situation, the information held in the coded data buffer is applied to the Huffman decoder and video demultiplexor as a total picture. In this way, the state machine of the Huffman decoder and video demultiplexor can still handle the data according to system design.

Another advantage of the PICTURE_END control token is its ability to completely empty the coded data buffer so that no stray information will inadvertently remain in the off chip DRAM or in the swing buffers.

Yet another advantage of the PICTURE_END function is its use in error recovery. For example, assume the amount of data being held in the coded data buffer is less than is typically used for describing the spatial information with reference to a single picture. Accordingly, the last picture will be held in the data buffer until a full swing buffer, but, by definition, the buffer will never fill. At some point, the machine will determine that an error condition exits. Hence, to the extent that a PICTURE_END token is decoded and forces the data in the coded data buffers to be applied to the Huffman decoder and video demultiplexor, the final picture can be decoded and the information emptied from the buffers. Consequently, the machine will not go into error recovery mode and will successfully continue to process the coded data.

A still further advantage of the use of a PICTURE_END token is that the serial pipeline processor will continue the processing of uninterrupted data. Through the use of a PICTURE_END token, the serial pipeline processor is configured to handle less than the expected amount of data and, therefore, continues processing. Typically, a prior token is not associated with either controlling the reconfiguration of the state machine or in providing data for the system. Rather, it completes prior partial signals for handling by the machine-dependent state machines. Each of the state machines recognizes a FLUSH control token as information not to be processed. Accordingly, the FLUSH token is used to fill up all of the remaining empty parts of the coded data buffers and to allow a full set of information to be sent to the Huffman Decoder and Video Demultiplexor. In this way, the FLUSH token is like padding for buffers.

The Token Decoder in the Huffman circuit recognizes the FLUSH token and ignores the pseudo data that the FLUSH token has forced into it. The Huffman Decoder then operates only on the data contents of the last picture buffer as it existed prior to the arrival of the PICTURE_END token and FLUSH token. A further advantage of the use of the PICTURE_END token alone or in combination with a FLUSH token is the reconfiguration and/or reorganization of the Huffman Decoder circuit. With the arrival of the PICTURE_END token, the Huffman Decoder circuit knows that it will have less information than normally expected to decode the last picture. The Huffman decode circuit finishes processing the information contained in the last picture, and outputs this information through the DRAM interface into the Inverse Modeller. Upon the identification of the last picture, the Huffman Decoder goes into its cleanup mode and readjusts for the arrival of the next picture information.

20. FLUSH FUNCTION

The FLUSH token, in accordance with the present invention, is used to pass through the entire pipeline processor and to ensure that the buffers are emptied and that other circuits are reconfigured to await the arrival th input to the serial pipeline processor to look at the incoming bit stream. When the search mode is set, the Start Code Detector searches only for a specific start code or marker used in any one of the compression standards. It will be appreciated, however, that, other images from other data bitstreams can be used for this purpose. Accordingly, these images can be used throughout this present invention to change it to another embodiment which is capable of using the combination of control tokens, and DATA tokens along with the reconfiguration circuits, to provide similar processing.

The use of search mode in the present invention is convenient in many situations including 1) if a break in the data bit stream occurs; 2) when the user breaks the data bit stream by purposely changing channels, e.g., data arriving, by a cable carrying compressed digital video; or 3) by user activation of fast forward or reverse from a controllable data source such as an optical disc or video disc. In general, a search mode is convenient when the user interrupts the normal processing of the serial pipeline at a point where the machine does not expect such an interruption.

When any of the search modes are set, the Start Code Detector looks for incoming start images which are suitable for creating the machine independent tokens. All data coming into the Start Code Detector prior to the identification of standard-dependent start images is discarded as meaningless and the machine stands in an idling condition as it waits this information.

The Start Code Detector can assume any one of a number of configurations. For example, one of these configurations allows a search for a group of pictures or higher start codes. This pattern causes the Start Code Detector to discard all its input and look for the

The Inverse Quantizer of the present invention is a required element in the decoding sequence, but has been implemented in such away to allow the entire IC set to handle multi-standard data. In addition, the Inverse Quantizer has been adapted for use with tokens. The Inverse Quantizer lies between the Inverse modeller and inverse DCT (IDCT).

For example, in the present invention, an adder in the Inverse Quantizer is used to add a constant to the pel decode number before the data moves on to the IDCT.

The IDCT uses the pel decode number, which will vary according to each standard used to encode the information. In order for the information to be properly decoded, a value of 1024 is added to the decode number by the Inverse Quantizer before the data continues on to the IDCT.

Using adders, already present in the Inverse Quantizer, to standardize the data prior to it reaching the IDCT, eliminates the need for additional circuitry or software in the IC, for handling data compressed by the various standards. Other operations allowing for multi-standard operation are performed during a "post quantization function" and are discussed below.

The control tokens accompanying the data are decoded and the various standardization routines that need to be performed by the Inverse Quantizer are identified in detail below. These "post quantization" functions are all implemented to avoid duplicate circuitry and to allow the IC to handle multi-standard encoded data.

25. HUFFMAN DECODER AND PARSER

Referring again to Figures 11 and 27, the Spatial Decoder includes a Huffman Decoder for decoding the data that the various compression standards have Huffman-encoded. While each of the standards, JPEG, MPEG and beneath the data line 328 that connects the blocks. Among other things, this code word identifies the particular standard that is being decoded.

The Huffman decoder 321 also performs certain control functions. In particular, the Huffman Decoder 321 contains a state machine that can control certain functions of the Index to Data 324 and ALU 325. Control of these units by the Huffman Decoder is necessary for proper decoding of block-level information. Having the Parser State Machine 322 make these decisions would take too much time.

An important aspect of the Huffman Decoder of the present invention, is the ability to invert the coded data bits as they are read into the Huffman Decoder. This is needed to decode H.261 style Huffman codes, since the particular type of Huffman code used by H.261 (and substantially by MPEG) has the opposite polarity then the codes used by JPEG. The use of an inverter, thereby, allows substantially the same table to be used by the Huffman Decoder for all three standards. Other aspects of how the Huffman Decoder implements all three standards are discussed in further detail in the "More Detailed Description of the Invention" section.

The Index to Data unit 324 performs the second part of the multi-part algorithm. This unit contains a look up table that provides the actual Huffman decoded data. Entries in the table are organized based on the index numbers generated by the Huffman Decoder.

The ALU 325 implements the remaining parts of the multi-part algorithm. In particular, the ALU handles sign-extension. The ALU also includes a register file which holds vector predictions and DC predictions, the use of which is described in the sections related to prediction filters. The ALU, further, includes counters that count through the structure of the picture being decoded by the format that JPEG specifies for transferring an alternate JPEG table.

From the Index to Data unit 324, the decoded index number or other data is passed, together with the accompanying control word, to the ALU 325, which performs the operations previously described.

From the ALU 325, the data and control word is passed to the Token Formatter 326 (TF). In the Token Formatter, the data is combined as needed with the control word to form tokens. The tokens are then conveyed to the next stages of the Spatial Decoder. Note that at this point, there are as many tokens as will be used by the system.

26. INVERSE DISCRETE COSINE TRANSFORM

The Inverse Discrete Cosine Transform (IDCT), in accordance with the present invention, decompresses data related to the frequency of the DC component of the picture. When a particular picture is being compressed, the frequency of the light in the picture is quantized, reducing the overall amount of information needed to be stored. The IDCT takes this quantized data and decompresses it back into frequency information.

The IDCT operates on a portion of the picture which is 8x8 pixels in size. The math which performed on this data is largely governed by the particular standard used to encode the data. However, in the present invention, significant use is made of common mathematical functions between the standards to avoid unnecessary duplication of circuitry.

Using a particular scaling order, the symmetry between the upper and lower portions of the algorithms is increased, thus common mathematical functions can be reused which eliminates the need for additional circuitry.

address generators.

The Buffer Manager also interfaces with the display address generators, receiving information on whether the display device is ready to display new data. The Buffer Manager also confirms that the display address generators have cleared information from a buffer for display.

The Buffer Manager of the present invention keeps track of whether a particular buffer is empty, full, ready for use or in use. It also keeps track of the presentation number associated with the particular data in each buffer. In this way, the Buffer Manager determines the states of the buffers, in part, by making only one buffer at a time ready for display. Once a buffer is displayed, the buffer is in a "vacant" state. When the Buffer Manager receives a PICTURE_START, FLUSH, valid or access token, it determines the status of each buffer and its readiness to accept new data. For example, the PICTURE_START token causes the Buffer Manager to cycle through each buffer to find one which is capable of accepting the new data.

The Buffer Manager can also be configured to handle the multi-standard requirements dictated by the tokens it receives. For example, in the H.261 standard, data maybe skipped during display. If such a token arrives at the Buffer Mnager, the data to be skipped will be flushed from the buffer in which it is stored.

Thus, by managing the buffers, data can be effectively displayed according to the compression standard used to encode the data, the rate at which the data is decoded and the particular type of display device being used.

This is a more detailed description for a multi-standard video decoder chip-set. It is divided into three main sections: A, B and C.

Again, for purposes of organization, clarity and convenience of explanation, this additional disclosure is set forth in the following sections.

- Description of features common to chips in the chip-set:
  - Tokens
  - Two wire interfaces
  - DRAM interface
  - Microprocessor interface
  - Clocks
  - Description of the Spatial Decoder chip
  - Description of the Temporal Decoder chip

SECTION A.1

The first description section covers the majority of the electrical design issues associated with using the chip-set.

A.1.1 Typographic conventions

A small set of typographic conventions is used to emphasize some classes of information:

NAMES_OF_TOKENS

wire_name active high signal

wire_name active low signal

register_name

re-format it for a computer or display system. The details of this formatting will vary between applications. In a simple case, all that is required is an address generator to take the block formatted data output by the decoder chip and write it into memory in a raster order.

The Image Formatter is a single chip VLSI device providing a wide range of output formatting functions.

A.2.1.2 JPEG still picture decoding

A single Spatial Decoder, with no-off-chip DRAM, can rapidly decode baseline JPEG images. The Spatial Decoder will support all features of baseline JPEG. However, the image size that can be decoded may be limited by the size of the output buffer provided by the user. The characteristics of the output formatter may limit the chroma sampling formats and color spaces that can be supported.

A.2.1.3 JPEG video decoding

Adding off-chip DRAMs to the Spatial Decoder allows it to decode JPEG encoded video pictures in real-time. The size and speed of the required buffers will depend on the video and coded data rates. The Temporal Decoder is not required to decode JPEG encoded video. However, if a Temporal Decoder is present in a multi-standard decoder chip-set, it will merely pass the data through the Temporal Decoder without alteration or modification when the system is configured for JPEG operation.

A.2.1.4 H.261 decoding

The Spatial Decoder and the Temporal Decoder are both required to implement an H.261 video decoder. The DRAM interfaces on both devices are configurable to allow the quantity of DRAM required for proper operation to be reduced when working with small picture formats and at low coded data rates. Typically, a single 4Mb (e.g. 512k x 8) DRAM will be required by each of the Spatial Decoder and

SECTION A.3 Tokens

A.3.1 Token format

In accordance with the present invention, tokens provide an extensible format for communicating information through the decoder chip-set. While in the present invention, each word of a Token is a minimum of 8 bits wide, one of ordinary skill in the art will appreciate that tokens can be of any width. Furthermore, a single Token can be spread over one or more words; this is accomplished using an extension bit in each word. The formats for the tokens are summarized in Table A.3.1.

The extension bit indicates whether a Token continues into another word. It is set to 1 in all words of a Token except the last one. If the first word of a Token has an extension bit of 0, this indicates that the Token is only one word long.

Each Token is identified by an Address Field that starts in bit 7 of the first word of the Token. The Address Field is of variable length and can potentially extend over multiple words (in the current chips no address is more than 8 bits long, however, one of ordinary skill in the art will again appreciate that addresses can be of any length).

Some interfaces transfer more than 8 bits of data. For example, the output of the Spatial Decoder is 9 bits wide (10 bits including the extension bit). The only Token that takes advantage of these extra bits is the DATA Token. The DATA Token can have as many bits as are necessary for carrying out processing at a particular place in the system. All other Tokens ignore the extra bits.

Furthermore, the data input to the Spatial Decoder can either be supplied as bytes of coded data, or in DATA Tokens (see Section A.10, "Coded data input"). Supplying Tokens via the coded data port or via the microprocessor interface allows many of the features of the decoder chip set to be configured from the data stream. This provides an alternative to doing the configuration via the micro processor interface.

7	6	5	4	3	2	1	0	Token Name	Rel rence
0	0	0	1	1	0	0	1	MAX_COMP_ID	
0	0	0	1	1	0	1	0	EXTENSION_DATA	
0	0	0	1	1	0	1	1	USER_DATA	
0	0	0	1	1	1	0	0	DHT_MARKER	
0	0	0	1	1	1	0	1	DQT_MARKER	
0	0	0	1	1	1	1	0	(reserved) DNL_MARKER	
0	0	0	1	1	1	1	1	(reserved) DRI_MARKER	
1	1	1	0	1	0	0	0	(reserved)	
1	1	1	0	1	0	0	1	(reserved)	
1	1	1	0	1	0	1	0	(reserved)	
1	1	1	0	1	0	1	1	(reserved)	
1	1	1	0	1	1	0	0	BIT_RATE	
1	1	1	0	1	1	0	1	VBV_BUFFER_SIZE	
1	1	1	0	1	1	1	0	VBV_DELAY	
1	1	1	0	1	1	1	1	PICTURE_TYPE	
1	1	1	1	0	0	0	0	PICTURE_RATE	
1	1	1	1	0	0	0	1	PEL_ASPECT	
1	1	1	1	0	0	1	0	HORIZONTAL_SIZE	
1	1	1	1	0	0	1	1	VERTICAL_SIZE	
1	1	1	1	0	1	0	0	BROKEN_CLOSED	
1	1	1	1	0	1	0	1	CONSTRAINED	
1	1	1	1	0	1	1	0	(reserved) SPECTRAL_LIMIT	
1	1	1	1	0	1	1	1	DEFINE_MAX_SAMPLING	
1	1	1	1	1	0	0	0	(reserved)	
1	1	1	1	1	0	0	1	(reserved)	
1	1	1	1	1	0	1	0	(reserved)	
1	1	1	1	1	0	1	1	(reserved)	
1	1	1	1	1	1	0	0	HORIZONTAL_MBS	
1	1	1	1	1	1	0	1	VERTICAL_MBS	
1	1	1	1	1	1	1	0	(reserved)	
1	1	1	1	1	1	1	1	(reserved)	

Table A.3.1 Summary of Tokens (contd)

E	7	6	5	4	3	2	1	0	Description
1	1	1	1	0	1	1	0	0	**BIT_RATE** test info only
1	r	r	r	r	r	r	b	b	Carries the MPEG bit rate parameter R. Generated by the Huffman
1	b	b	b	b	b	b	b	b	decoder when decoding an MPEG bitstream.
0	b	b	b	b	b	b	b	b	b - an 18 bit integer as defined by MPEG
1	1	1	1	1	0	1	0	0	**BROKEN_CLOSED**
0	r	r	r	r	r	r	c	b	Carries two MPEG flags bits: c - closed_gop b - broken_link
1	0	0	0	1	0	1	0	1	**CODING_STANDARD**
0	s	s	s	s	s	s	s	s	s - an 8 bit integer indicating the current coding standard. The values currently assigned are: 0 - H.261 1 - JPEG 2 - MPEG
1	1	1	0	0	0	0	c	c	**COMPONENT_NAME**
0	n	n	n	n	n	n	n	n	Communicates the relationship between a component ID and the component name. See also ... c - 2 bit component ID n - 8 bit component "name"
1	1	1	1	1	0	1	0	1	**CONSTRAINED**
0	r	r	r	r	r	r	r	c	c - carries the constrained_parameters_flag decoded from an MPEG bitstream.

Table A.3.2 Tokens implemented in the Spatial Decoder and Temporal Decoder (Sheet 1 of 9)

E	7	6	5	4	3	2	1	0	Description
0	0	0	0	1	1	1	1	0	**DNL_MARKER** This Token informs the Video Demux that the DATA Token that follows contains the JPEG parameter NL which specifies the number of lines in a frame. This Token is generated by the start code detector during JPEG decoding when a DNL marker has been encountered in the data stream.
0	0	0	0	1	1	1	0	1	**DQT_MARKER** This Token informs the Video Demux that the DATA Token that follows contains the specification of a quantisation table described using the JPEG "define quantisation table segment" syntax. This Token is only valid when the coding standard is configured as JPEG. The Video Demux generates a QUANT_TABLE Token containing the new quantisation table information. This Token is generated by the start code detector during JPEG decoding when a DQT marker has been encountered in the data stream.
0	0	0	0	1	1	1	1	1	**DRI_MARKER** This Token informs the Video Demux that the DATA Token that follows contains the JPEG parameter Ri which specifies the number of minimum coding units between restart markers. This Token is generated by the start code detector during JPEG decoding when a DRI marker has been encountered in the data stream.

Table A.3.2 Tokens implemented in the Spatial Decoder and Temporal Decoder (Sheet 3 of 9)

E	7	6	5	4	3	2	1	0	Description
1	1	1	1	1	1	1	0	0	**HORIZONTAL_MBS**
1	r	r	r	h	h	h	h	h	h - a 13 bit number integer indicating the horizontal width of the
0	h	h	h	h	h	h	h	h	picture in macroblocks.
1	1	1	1	1	0	0	1	0	**HORIZONTAL_SIZE**
1	h	h	h	h	h	h	h	h	h - 16 bit number integer indicating the horizontal width of the
0	h	h	h	h	h	h	h	h	picture in pixels. This can be any integer value.
1	1	1	0	0	1	0	c	c	**JPEG_TABLE_SELECT**
0	r	r	r	r	r	r	t	t	Informs the inverse quantiser which quantisation table to use on the specified colour component. c - 2 bit component ID (see A.3.5.1 t - 2 bit integer table number.
1	0	0	0	1	1	0	0	1	**MAX_COMP_ID**
0	r	r	r	r	r	r	m	m	m - 2 bit integer indicating the maximum value of component ID (see A.3.5.1) that will be used in the next picture.
0	1	1	0	1	0	1	c	c	**MPEG_DCH_TABLE**
0	r	r	r	r	r	r	t	t	Configures which DC coefficient Huffman table should be used for colour component cc. c - 2 bit component ID (see A.3.5.1 t - 2 bit integer table number.
0	1	1	0	0	1	1	d	n	**MPEG_TABLE_SELECT** Informs the inverse quantiser whether to use the default or user defined quantisation table for intra or non-intra information. n - 0 indicates intra information, 1 non-intra. d - 0 indicates default table, 1 user defined.

Table A.3.2 Tokens implemented in the Spatial Decoder and Temporal Decoder (Sheet 5 of 9)

E	7	6	5	4	3	2	1	0	Description
1	1	1	1	0	1	1	1	1	PICTURE_TYPE MPEG
0	r	r	r	r	r	r	p	p	p - a 2 bit integer indicating the picture coding type of the picture that follows: 0 - Intra 1 - Predicted 2 - Bidirectionally Predicted 3 - DC Intra
1	1	1	1	0	1	1	1	1	PICTURE_TYPE H.261
1	r	r	r	r	r	r	0	1	Indicates various H.261 options are on (1) or off (0). These options are always off for MPEG and JPEG:
0	r	r	s	d	f	q	1	1	s - Split Screen Indicator d - Document Camera f - Freeze Picture Release Source picture format: q = 0 - QCIF q = 1 - CIF
0	0	1	0	h	y	x	b	f	PREDICTION_MODE A set of flag bits that indicate the prediction mode for the macroblocks that follow: f - forward prediction b - backward prediction x - reset forward vector predictor y - reset backward vector predictor h - enable H.261 loop filter
0	0	0	1	s	s	s	s	s	QUANT_SCALE Informs the inverse quantiser of a new scale factor s - 5 bit integer in range 1 ... 31. The value 0 is reserved.

Table A.3.2 Tokens implemented in the Spatial Decoder and Temporal Decoder (Sheet 7 of

E	7	6	5	4	3	2	1	0	Description
1	0	0	0	1	1	0	1	1	**USER_DATA JPEG**
0	v	v	v	v	v	v	v	v	This Token informs the Video Demux that the DATA Token that follows contains user data. See A.11.3, "Conversion of start codes to Tokens", and A.14.6, "Receiving User and Extension data". During JPEG operation the 8 bit field "v" carries the JPEG marker value. This allows the class of user data to be identified.
0	0	0	0	1	1	0	1	1	**USER_DATA MPEG** This Token informs the Video Demux that the DATA Token that follows contains user data. See A.11.3, "Conversion of start codes to Tokens", and A.14.6, "Receiving User and Extension data".
1	1	1	1	0	1	1	0	1	**VBV_BUFFER_SIZE**
1	r	r	r	r	r	r	s	s	s - a 10 bit integer as defined by MPEG.
0	s	s	s	s	s	s	s	s	
1	1	1	1	0	1	1	1	0	**VBV_DELAY**
1	b	b	b	b	b	b	b	b	b - a 16 bit integer as defined by MPEG.
0	b	b	b	b	b	b	b	b	
1	1	1	1	1	1	1	0	1	**VERTICAL_MBS**
1	r	r	r	v	v	v	v	v	v - a 13 bit integer indicating the vertical size of the picture in macroblocks.
0	v	v	v	v	v	v	v	v	
1	1	1	1	1	0	0	1	1	**VERTICAL_SIZE**
1	v	v	v	v	v	v	v	v	v - a 16 bit integer indicating the vertical size of the picture in pixels. This can be any integer value.
0	v	v	v	v	v	v	v	v	

Table A.3.2 Tokens implemented in the Spatial Decoder and Temporal Decoder (Sheet 9 of 9)

With JPEG the situation is more complex as JPEG does not limit the color components that can be used. The decoder chips permit up to 4 different color components in each scan. The IDs are allocated sequentially as the specification of color components arrive at the decoder.

A.3.5.2 Horizontal and Vertical sampling numbers

For each of the 4 color components, there is a specification for the number of blocks arranged horizontally and vertically in a macroblock. This specification comprises a two bit integer which is one less than the number of blocks.

For example, in MPEG (or H.261) with 4:2:0 chroma sampling (Figure 36) and component IDs allocated as per Table A.3.4.

Component ID	Horizontal sampling number	Width in blocks	Vertical sampling number	Height in blocks
0	1	2	1	2
1	0	1	0	1
2	0	1	0	1
3	Not used	Not used	Not used	Not used

Table A.3.4 Sampling numbers for 4:2:0/MPEG

A.3.6 Special Token formats

In accordance with the present invention, tokens such as the DATA Token and the QUANT_TABLE Token are used in their "extended form" within the decoder chip-set. In the extended form the Token includes some data. In the case of DATA Tokens, they can contain coded data or pixel data. In the case of QUANT_TABLE tokens, they contain quantizer table information.

Furthermore, "non-extended form" of these Tokens is defined in the present invention as "empty". This Token format provides a place in the Token stream that can be subsequently filled by an extended version of the same Token. This format is mainly applicable to encoders and, therefore, it is not documented further here.

Token Name	MPEG	JPEG	H.261
BIT_RATE	✓		
BROKEN_CLOSED	✓		
CODING_STANDARD	✓	✓	✓
COMPONENT_NAME		✓	
CONSTRAINED	✓		
DATA	✓	✓	✓
DEFINE_MAX_SAMPLING	✓	✓	✓
DEFINE_SAMPLING	✓	✓	✓
DHT_MARKER		✓	
DNL_MARKER		✓	
DQT_MARKER		✓	
DRI_MARKER		✓	

Table A.3.6 tokens for different standards

A.3.7 Use of Tokens for different standards

Each standard uses a different sub-set of the defined Tokens in accordance with the present invention; ss Table A.3.6.

Interface	Data Width (bits)
Coded data input to Spatial Decoder	8
Output port of Spatial Decoder	9
Input port of Temporal Decoder	9
Output port of Temporal Decoder	8
Input port of Image Formatter	8

Table A.4.1 Two wire interface data width

In addition to the data signals there are three other signals transmitted via the two-wire interface:

.valid

.accept

.extension

A.4.3.1 The extension signal

The extension signal corresponds to the Token extension bit previously described.

A.4.4 Design considerations

The two wire interface is intended for short range, point to point communication between chips.

The decoder chips should be placed adjacent to each other, so as to minimize the length of the PCB tracks between chips. Where possible, track lengths should be kept below 25 mm. The PCB track capacitance should be kept to a minimum.

A.4.6 Signal l vels

The two-wire interface uses CMOS inputs and output. V_{IHmin} is approx. 70% of V_{DD} and V_{ILmax} is approx. 30% of V_{DD}. The values shown in Table A.4.3 are those for V_{IH} and V_{IL} at their respective worst case V_{DD}. $V_{DD}=5.0\pm0.25V$.

Symbol	Parameter	Min.	Max.	Units
V_{IH}	Input logic '1' voltage	3.68	$V_{DD}+0.5$	V
V_{IL}	Input logic '0' voltage	GND - 0.5	1.43	V
V_{OH}	Output logic '1' voltage	$V_{DD}-0.1$		V^{a}
		$V_{DD}-0.4$		V^{b}
V_{OL}	Output logic '0' voltage		0.1	V^{c}
			0.4	V^{d}
I_{IN}	Input leakage current		± 10	μA

Table A.4.3 DC electrical characteristics

a. $I_{OH}\leq 1mA$
b. $I_{OH}\leq 4mA$
c. $I_{OL}\leq 1mA$
d. $I_{OL}\leq 4mA$

SECTION A.5 DRAM Interface

A.5.1 The DRAM interfac

A single high performance, configurable, DRAM interface is used on each of the video decoder chips. In general, the DRAM interface on each chip is substantially the same; however, the interfaces differ from one another in how they handle channel priorities. The interface is designed to directly drive the DRAM used by each of the decoder chips. Typically, no external logic, buffers or components will be necessary to connect the DRAM interface to the DRAMs in most systems.

A.5.2 Interface signals

Signal Name	Input / Output	Description
DRAM_data[31:0]	I/O	The 32 bit wide DRAM data bus. Optionally this bus can be configured to be 16 or 8 bits wide. See section A.5.8
DRAM_addr[10:0]	O	The 22 bit wide DRAM interface address is time multiplexed over this 11 bit wide bus.
$\overline{\text{RAS}}$	O	The DRAM Row Address Strobe signal
$\overline{\text{CAS}}$[3:0]	O	The DRAM Column Address Strobe signal. One signal is provided per byte of the interface's data bus. All the $\overline{\text{CAS}}$ signals are driven simultaneously.
$\overline{\text{WE}}$	O	The DRAM Write Enable signal
$\overline{\text{OE}}$	O	The DRAM Output Enable signal
DRAM_enable	I	This input signal, when low, makes all the output signals on the interface go high impedance. Note: on-chip data processing is not stopped when the DRAM interface is high impedance. So, errors will occur if the chip attempts to access DRAM while DRAM_enable is low.

Table A.5.1 DRAM interface signals

A.5.3.3 Interface timing configuration

In accordance with the present invention, modifications to the interface timing configuration information are controlled by the interface_timing_access register. Writing 1 to this register allows the interface timing registers (in Table A.5.2) to be modified. While interface_timing_access = 1, the DRAM interface continues operation with its previous configuration. After writing 1, the user should wait until 1 can be read back from the interface_timing_access before writing to any of the interface timing registers.

When configuration is compete, 0 should be written to the interface_timing_access. The new configuration will then be transferred to the DRAM interface.

A.5.3.4 Refresh configuration

The refresh interval of the DRAM interface of the present invention can only be configured once following reset. Until refresh_interval is configured, the interface continually executes refresh cycles. This prevents any other data transfers. Data transfers can start after a value is written to refresh_interval.

As is well known in the art, DRAMs typically require a "pause" of between 100 μs and 500 μs after power is first applied, followed by a number of refresh cycles before normal operation is possible. Accordingly, these DRAM start-up requirements should be satisfied before writing a value to refresh_interval.

A.5.3.5 Read access to configuration registers

All the DRAM interface registers of the present invention can be read at any time.

A.5.4 Interface timing (ticks)

A.5.5 Interface r gist rs

Register name	Size/Dir.	Reset State	Description
interface_timing_access	1 bit rw	0	This function enable register allows access to the DRAM interface timing configuration registers. The configuration registers should not be modified while this register holds the value 0. Writing a one to this register requests access to modify the configuration registers. After a 0 has been written to this register the DRAM interface will start to use the new values in the timing configuration registers.
page_start_length	5 bit rw	0	Specifies the length of the access start in ticks. The minimum value that can be used is 4 (meaning 4 ticks). 0 selects the maximum length of 32 ticks.
transfer_cycle_length	4 bit rw	0	Specifies the length of the fast page read or write cycle in ticks. The minimum value that can be used is 4 (meaning 4 ticks). 0 selects the maximum length of 16 ticks.
refresh_cycle_length	4 bit rw	0	Specifies the length of the refresh cycle in ticks. The minimum value that can be used is 4 (meaning 4 ticks). 0 selects the maximum length of 16 ticks.
RAS_falling	4 bit rw	0	Specifies the number of ticks after the start of the access start that $\overline{RAS}$ falls. The minimum value that can be used is 4 (meaning 4 ticks). 0 selects the maximum length of 16 ticks.
CAS_falling	4 bit rw	8	Specifies the number of ticks after the start of a read cycle, write cycle or access start that $\overline{CAS}$ falls. The minimum value that can be used is 1 (meaning 1 tick). 0 selects the maximum length of 16 ticks.

Tabl A.5.2 Int rfac timing configuration registers

A.5.6 ~~-~~Interface operati n

The DRAM interface uses fast page mode. Three different types of access are supported:

.Read

.Write

.Refresh

Each read or write access transfers a burst of 1 to 64 bytes to a single DRAM page address. Read and write transfers are not mixed within a single access and each successive access is treated as a random access to a new DRAM page.

Register name	Size/Dir.	Reset State	Description
refresh_interval	8 bit rw	0	This value specifies the interval between refresh cycles in periods of 16 decoder_clock cycles. Values in the range 1..255 can be configured. The value 0 is automatically loaded after reset and forces the DRAM interface to continuously execute refresh cycles until a valid refresh interval is configured. It is recommended that refresh_interval should be configured *only once* after each reset.
no_refresh	1 bit rw	0	Writing the value 1 to this register prevents execution of any refresh cycles.

Table A.5.4 Refresh configuration registers

begin when the last access has finished, then the new access will begin immediately.

A.5.7.1 Access start

The *access start* provides the page address for the read or write transfers and establishes some initial signal conditions. In accordance with the present invention, there are three different access starts:

.Start of read

.Start of write

.Start of refresh

Num.	Characteristic	Min.	Max.	Unit	Notes
5	$\overline{\text{RAS}}$ precharge period set by register **RAS_falling**	4	16	bck	
6	Access start duration set by register **page_start_length**	4	32		
7	$\overline{\text{CAS}}$ precharge length set by register **CAS_falling.**	1	16		a
8	Fast page read or write cycle length set by the register **transfer_cycle_length.**	4	16		
9	Refresh cycle length set by the register **refresh_cycle.**	4	16		

Table A.5.5 DRAM Interface timing parameters

a. This value must be less than RAS_falling to ensure $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ refresh occurs.

The ~~two~~ bit register, DRAM_data_width, allows the width of the DRAM interface's data path to be configured. This allows the DRAM cost to be minimized when working with small picture formats.

DRAM_data_width	
0[a]	8 bit wide data bus on DRAM_data[31:24][b].
1	16 bit wide data bus on DRAM_data[31:16][b].
2	32 bit wide data bus on DRAM_data[31:0].

Table A.5.6 Configuring DRAM_data_width

a. Default after reset.
b. Unused signals are held high impedance.

A.5.9 row address width

The number of bits that are taken from the middle section of the 24 bit internal address in order to provide the row address is configured by the register, row_address_bits.

row_address_bits	Width of row address
1	10 bits on DRAM_addr[9:0]
2	11 bits on DRAM_addr[10:0]

Table A.5.7 Configuring row_address_bits

A.5.10.1 Low order column address bits

The least significant 4 to 6 bits of the column address are used to provide addresses for fast page mode transfers of up to 64 bytes. The number of address bits required to control these transfers will depend on the width of the data bus (see A.5.8).

A.5.10.2 Decoding row address to access more DRAM banks

Where only a single bank of DRAM is used, the width of the row address used will depend on the type of DRAM used. Applications that require more memory than can be typically provided by a single DRAM bank, can configure a wider row address and then decode some row address bits to select a single DRAM bank.

NOTE: The row address is extracted from the middle of the internal address. If some bits of the row address are decoded to select banks of DRAM, then all possible values of these "bank select bits" must select a bank of DRAM. Otherwise, holes will be left in the address space.

A.5.11 DRAM Interface enable

In the present invention, there are two ways to make all the output signals on the DRAM interface become high impedance, i.e., by setting the DRAM_enable register and the DRAM-enable signal. Both the register and the signal must be at a logic 1 in order for the drivers on the DRAM interface to operate. If either is low then the interface is taken to high impedance.

Note: on-chip data processing is not terminated when the DRAM interface is at high impedance. Therefore, errors will occur if the chip attempts to access DRAM while the interface is at high impedance.

In accordance with the present invention, the ability to take the DRAM interface to high impedance is provided to allow other devices to test or use the DRAM controlled by the Spatial Decoder (or the Temporal Decoder) when the

strength value	Drive characteristics
0	Approx. 4 ns/V into 6 pf load
1	Approx. 4 ns/V into 12 pf load
2	Approx. 4 ns/V into 24 pf load
3	Approx. 4 ns/V into 48 pf load
4	Approx. 2 ns/V into 6 pf load
5	Approx. 2 ns/V into 12 pf load
6[a]	Approx. 2 ns/V into 24 pf load
7	Approx. 2 ns/V into 48 pf load

Table A.5.9 Output strength configurations

a. Default after reset

When an output is configured appropriately for the load it is driving, it will meet the AC electrical characteristics specified in Tables A.5.13 to A.5.16. When appropriately configured, each output is approximately matched to its load and, therefore, minimal overshoot will occur after a signal transition.

A.5.14 Electrical specifications

All information provided in this section is merely illustrative of one embodiment of the present invention and is included by example and not necessarily by way of limitation.

Symbol	Parameter	Min.	Max.	Units
V_{OL}	Output logic '0' voltage		0.4	V [a]
V_{OH}	Output logic '1' voltage	2.8		V
I_O	Output current	± 100		µA [b]
I_{OZ}	Output off state leakage current	± 20		µA
I_{IZ}	Input leakage current	± 10		µA
I_{DD}	RMS power supply current		500	mA
C_{IN}	Input capacitance		5	pF
C_{OUT}	Output / IO capacitance		5	pF

Table A.5.12 DC Electrical characteristics

a. AC parameters are specified using V_{OLmax} = 0.8V as the measurement level.

b. This is the steady state drive capability of the interface.
Transient currents may be much greater.

Num.	Parameter	Min.	Max.	Unit	Note[a]
19	Set up time	-12	+3	ns	
20	Hold time	-12	+3	ns	
21	Address access time	-12	+3	ns	
22	Next valid after strobe	-12	+3	ns	

Table A.5.15 Differences from nominal between a bus and a strobe

a. The driver strength of the bus and the strobe must be configured appropriately for their loads.

Num.	Parameter	Min.	Max.	Unit	Note
23	Read data set-up time before $\overline{\text{CAS}}$ signal starts to rise	0		ns	
24	Read data hold time after $\overline{\text{CAS}}$ signal starts to go high	0		ns	

Table A.5.16 Differences from nominal between a bus and a strobe

When reading from DRAM, the DRAM interface samples DRAM_data[31:0] as the $\overline{\text{CAS}}$ signals rise.

SECTION A.6 Microprocessor interface (MPI)

A standard byte wide microprocessor interface (MPI) is used on all chips in the video decoder chip-set. However, one of ordinary skill in the art will appreciate that microprocessor interfaces of other widths may also be used. The MPI operates synchronously to various decoder chip clocks.

A.6.1 MPI signals

Signal Name	Input / Output	Description
$\overline{\text{enable}}$[1:0]	Input	Two active low chip enables. Both must be low to enable accesses via the MPI.
r$\overline{\text{w}}$	Input	High indicates that a device wishes to read values from the video chip. This signal should be stable while the chip is enabled.
addr[n:0]	Input	Address specifies one of 2^n locations in the chip's memory map. This signal should be stable while the chip is enabled.
data[7:0]	Output	8 bit wide data I/O port. These pins are high impedance if either enable signal is high.
$\overline{\text{irq}}$	Output	An active low, open collector, interrupt request signal.

Table A.6.1 MPI interface signals

Symbol	Parameter	Min.	Max.	Units
V_{OL}	Output logic '0' voltage		0.4	V
V_{OLoc}	Open collector output logic '0' voltage		0.4	V[a]
V_{OH}	Output logic '1' voltage	2.4		V
I_O	Output current	± 100		µA[b]
I_{Ooc}	Open collector output current	4.0	8.0	mA[c]
I_{OZ}	Output off state leakage current		± 20	µA
I_{IN}	Input leakage current		± 10	µA
I_{DD}	RMS power supply current		500	mA
C_{IN}	Input capacitance		5	pF
C_{OUT}	Output / IO capacitance		5	pF

Table A.6.4 DC Electrical characteristics

a. $I_O \le I_{Ooc\ min}$.

b. This is the steady state drive capability of the interface. Transient currents may be much greater.

c. When asserted the open collector $\overline{irq}$ output pulls down with an impedance of 100Ω or less.

A.6.3 Interrupts

In accordance with the present invention, "event" is the term used to describe an on-chip condition that a user might want to observe. An event can indicate an error or it can be informative to the user's software.

There are two single bit registers associated with each interrupt or "event". These are the *condition event register* and the *condition mask register*.

A.6.3.1 condition event register

The condition event register is a one bit read/write register whose value is set to one by a condition occurring within the circuit. The register is set to one even if the condition was merely transient and has now gone away. The register is then guaranteed to remain set to one until the user's software resets it (or the entire chip is reset).

- The register is set to zero by writing the value one
- Writing zero to the register leaves the register unaltered.
- The register must be set to zero by user software
  before another occurrence of this condition can be observed.
- The register will be reset to zero on reset.

A.6.3.2 Condition mask register

The condition mask register is one bit read/write register which enables the generation of an interrupt request if the corresponding condition event register(s) is(are) set. If the condition event is already set when 1 is written to the condition mask register, an interrupt request will be issued immediately.

- The value 1 enables interrupts.
- The register clears to zero on reset.

Unless stated otherwise a block will stop operation modified if the block with which they are associated is stopped. Therefore, groups of registers will normally be associated with an *access register*.

The value 0 in an access register indicates that the group of registers associated with that access register should not be modified. Writing 1 to an access register requests that a block be stopped. However, the block may not stop immediately and block's access register will hold the value 0 until it is stopped.

Accordingly, user software should wait (after writing 1 to request access) until 1 is read from the access register. If the user writes a value to a configuration register while its access register is set to 0, the results are undefined.

A.6.4.2 Registers holding integers

The least significant bit of any byte in the memory map is that associated with the signal data[0].

Registers that hold integers values greater than 8 bits are split over either 2 or 4 consecutive byte locations in the memory map. The byte ordering is "big endian" as shown in Figure 55. However, no assumptions are made about the order in which bytes are written into multi-byte registers.

Unused bits in the memory map will return a 0 when read except for unused bits in registers holding signed integers. In this case, the most significant bit of the register will be sign extended. For example, a 12 bit *signed* register will be sign extended to fill a 16 bit memory map location (two bytes). A 16 bit memory map location holding a 12 bit *unsigned* integer will return a 0 from its most significant bits.

A.6.4.3 Keyholed address locations

In the present invention, certain less frequently accessed memory map locations have been placed behind testability. Therefore, these registers have no application in the normal use of the devices and need not be accessed by normal device configuration and control software.

A.7.2 —Temp ral Decod r clock signals

The Temporal Decoder has only one clock input:

Signal Name	Input / Output	Description
decoder_clock	Input	The decoder clock controls all of the processing functions on the Temporal Decoder. The decoder clock also controls transfer of data in to the Temporal Decoder through its input port and out via its output port.

Table A.7.2 Temporal Decoder clocks

A.7.3 Electrical specifications

Num.	Characteristic	30 MHz		Unit	Note
		Min.	Max.		
35	Clock period	33		ns	
36	Clock high period	13		ns	
37	Clock low period	13		ns	

5

Table A.7.3 Input clock requirements

SECTION A.8 JTAG

As circuit boards become more densely populated, it is increasingly difficult to verify the connections between components by traditional means, such as in-circuit testing using a bed-of-nails approach. In an attempt to resolve the access problem and standardize on a methodology, the Joint Test Action Group (JTAG) was formed. The work of this group culminated in the "Standard Test Access Port and Boundary Scan Architecture", now adopted by the IEEE as standard 1149.1. The Spatial Decoder and Temporal Decoder comply with this standard.

The standard utilizes a boundary scan chain which serially connects each digital signal pin on the device. The test circuitry is transparent in normal operation, but in test mode the boundary scan chain allows test patterns to be shifted in, and applied to the pins of the device. The resultant signals appearing on the circuit board at the inputs to the JTAG device, may be scanned out and checked by relatively simple test equipment. By this means, the inter-component connections can be tested, as can areas of logic on the circuit board.

All JTAG operations are performed via the Test Access Port (TAP), which consists of five pins. The $\overline{\text{trst}}$ (Test Reset) pin resets the JTAG circuitry, to ensure that the device doesn't power-up in test mode. The tck (Test Clock) pin is used to clock serial test patterns into the tdi (Test Data Input) pin, and out of the tdo (Test Data Output) pin. Lastly, the operational mode of the JTAG circuitry is set by clocking the appropriate sequence of bits into the tms (Test Mode Select) pin.

The JTAG standard is extensible to provide for additional features at the discretion of the chip manufacturer. On the Spatial Decoder and Temporal Decoder,

A.8.2 Level of Conformance to IEEE 1149.1

A.8.2.1 Rules

All rules are adhered to, although the following should be noted:

Rules	Description
3.1.1(b)	The $\overline{trst}$ pin is provided.
3.5.1(b)	Guaranteed for all public instructions (see IEEE 1149.1 5.2.1(c)).
5.2.1(c)	Guaranteed for all public instructions. For some private instructions, the TDO pin may be active during any of the states Capture-DR, Exit1-DR, Exit-2-DR & Pause-DR.
5.3.1(a)	Power on-reset is achieved by use of the $\overline{trst}$ pin.
6.2.1(e,f)	A code for the BYPASS instruction is loaded in the Test-Logic-Reset state.
7.1.1(d)	Un-allocated instruction codes are equivalent to BYPASS.
7.2.1(c)	There is no device ID register.

5

Table A.8.2 JTAG Rules

Recommendation	Description
10.4.1(f)	During EXTEST, the signal driven into the on-chip logic from the system clock pin is that supplied externally.

Table A.8.4 Recommendations not implemented

A.8.2.3 Permissions

Permissions	Description
3.2.1(c)	Guaranteed for all public instructions.
6.1.1(f)	The instruction register is not used to capture design-specific information.
7.2.1(g)	Several additional public instructions are provided.
7.3.1(a)	Several private instruction codes are allocated.
7.3.1(c)	(Rule?) Such instructions codes are documented.
7.4.1(f)	Additional codes perform identically to BYPASS.
10.1.1(i)	Each output pin has its own 3-state control.
10.3.1(h)	A parallel latch is provided.
10.3.1(i,j)	During EXTEST, input pins are controlled by data shifted in via tdl.
10.6.1(d,e)	3-state cells are not forced inactive in the Test-Logic-Reset state.

Table A.8.5 Permissions met

A.9.1 Spatial D coder Signals

Signal Name	I/O	Pin Number	Description
coded_clock	I	182	Coded Data Port. Used to supply coded data or Tokens to the Spatial Decoder. See sections A.10.1 and A.4.1
coded_data[7:0]	I	172, 171, 169, 168, 167, 166, 164, 163	
coded_extn	I	174	
coded_valid	I	162	
coded_accept	O	161	
byte_mode	I	176	
$\overline{\text{enable}}$[1:0]	I	126, 127	Micro Processor Interface (MPI). See section A.6.1
r$\overline{\text{w}}$	I	125	
addr[6:0]	I	136, 135, 133, 132, 131, 130, 128	
data[7:0]	O	152, 151, 149, 147, 145, 143, 141, 140	
$\overline{\text{irq}}$	O	154	
DRAM_data[31:0]	I/O	15, 17, 19, 20, 22, 25, 27, 30, 31, 33, 35, 38, 39, 42, 44, 47, 49, 57, 59, 61, 63, 66, 68, 70, 72, 74, 76, 79, 81, 83, 84, 85	DRAM Interface. See section A.5.2
DRAM_addr[10:0]	O	184, 186, 188, 189, 192, 193, 195, 197, 199, 200, 203	
$\overline{\text{RAS}}$	O	11	
$\overline{\text{CAS}}$[3:0]	O	2, 4, 6, 8	
$\overline{\text{WE}}$	O	12	
$\overline{\text{OE}}$	O	204	
DRAM_enable	I	112	
out_data[8:0]	O	88, 89, 90, 92, 93, 94, 95, 97, 98	Output Port. See section A.4.1
out_extn	O	87	
out_valid	O	99	
out_accept	I	100	
tck	I	115	JTAG port. See section A.8
tdi	I	116	
tdo	O	120	
tms	I	117	
$\overline{\text{trst}}$	I	121	

Table A.9.1 Spatial Decoder signals

Signal Name	Pin	Signal Name	Pin	Signal Name	Pin	Signal Name	Pin
nc	208	nc	156	nc	104	nc	52
test pin	207	nc	155	nc	103	nc	51
test pin	206	$\overline{\text{irq}}$	154	nc	102	nc	50
GND	205	nc	153	VDD	101	DRAM_data(15)	49
OE	204	data(7)	152	out_accept	100	nc	48
DRAM_addr(0)	203	data(6)	151	out_valid	99	DRAM_data(16)	47
VDD	202	nc	150	out_data(0)	98	nc	46
nc	201	data(5)	149	out_data(1)	97	GND	45
DRAM_addr(1)	200	nc	148	GND	96	DRAM_data(17)	44
DRAM_addr(2)	199	data(4)	147	out_data(2)	95	nc	43
GND	198	GND	146	out_data(3)	94	DRAM_data(18)	42
DRAM_addr(3)	197	data(3)	145	out_data(4)	93	VDD	41
nc	196	nc	144	out_data(5)	92	nc	40
DRAM_addr(4)	195	data(2)	143	VDD	91	DRAM_data(19)	39
VDD	194	nc	142	out_data(6)	90	DRAM_data(20)	38
DRAM_addr(5)	193	data(1)	141	out_data(7)	89	nc	37
DRAM_addr(6)	192	data(0)	140	out_data(8)	88	GND	36
nc	191	nc	139	out_extn	87	DRAM_data(21)	35
GND	190	VDD	138	GND	86	nc	34
DRAM_addr(7)	189	nc	137	DRAM_data(0)	85	DRAM_data(22)	33
DRAM_addr(8)	188	addr(6)	136	DRAM_data(1)	84	VDD	32
VDD	187	addr(5)	135	DRAM_data(2)	83	DRAM_data(23)	31
DRAM_addr(9)	186	GND	134	VDD	82	DRAM_data(24)	30
nc	185	addr(4)	133	DRAM_data(3)	81	nc	29
DRAM_addr(10)	184	addr(3)	132	nc	80	GND	28
GND	183	addr(2)	131	DRAM_data(4)	79	DRAM_data(25)	27
coded_clock	182	addr(1)	130	GND	78	nc	25
VDD	181	VDD	129	nc	77	DRAM_data(26)	25
test pin	180	addr(0)	128	DRAM_data(5)	76	nc	24
test pin	179	$\overline{\text{enable}}$(0)	127	nc	75	VDD	23
test pin	178	$\overline{\text{enable}}$(1)	126	DRAM_data(6)	74	DRAM_data(27)	22
decoder_clock	177	$\overline{\text{rw}}$	125	VDD	73	nc	21
byte_mode	176	GND	124	DRAM_data(7)	72	DRAM_data(28)	20
GND	175	test pin	123	nc	71	DRAM_data(29)	19
coded_extn	174	test pin	122	DRAM_data(8)	70	GND	18

Table A.9.3 Spatial Decoder Pin Assignments

Signal Name	Pin	Signal Name	Pin	Signal Name	Pin	Signal Name	Pin
nc	173	$\overline{\text{trst}}$	121	GND	69	DRAM_data[30]	17
coded_data[7]	172	tdo	120	DRAM_data[9]	68	nc	16
coded_data[6]	171	nc	119	nc	67	DRAM_data[31]	15
VDD	170	VDD	118	DRAM_data[10]	66	VDD	14
coded_data[5]	169	tms	117	VDD	65	nc	13
coded_data[4]	168	tdi	116	nc	64	$\overline{\text{WE}}$	12
coded_data[3]	167	tck	115	DRAM_data[11]	63	$\overline{\text{RAS}}$	11
coded_data[2]	166	test pin	114	nc	62	nc	10
GND	165	GND	113	DRAM_data[12]	61	GND	9
coded_data[1]	164	DRAM_enable	112	GND	60	$\overline{\text{CAS}}$[0]	8
coded_data[0]	163	test pin	111	DRAM_data[13]	59	nc	7
coded_valid	162	test pin	110	nc	58	$\overline{\text{CAS}}$[1]	6
coded_accept	161	test pin	109	DRAM_data[14]	57	VDD	5
reset	160	nc	108	VDD	56	$\overline{\text{CAS}}$[2]	4
VDD	159	nc	107	nc	55	nc	3
nc	158	nc	106	nc	54	$\overline{\text{CAS}}$[3]	2
nc	157	nc	105	nc	53	nc	1

Table A.9.3 Spatial Decoder Pin Assignments (contd)

A.9.1.1 "nc" no connect pins

The pins labeled nc in Table A.9.3 are not currently used these pins should be left unconnected.

A.9.1.2 V_{DD} and GND pins

As will be appreciated by one of ordinary skill in the art, all the V_{DD} and GND pins provided should be connected to the appropriate power supply. Correct device operation

A.9.2 Spatial Decoder memory map

Addr. (hex)	Register Name	See table
0x00 ... 0x03	Interrupt service area	A.9.6
0x04 ... 0x07	Input circuit registers	A.9.7
0x08 ... 0x0F	Start code detector registers	
0x10 ... 0x15	Buffer start-up control registers	A.9.8
0x16 ... 0x17	Not used	
0x18 ... 0x23	DRAM interface configuration registers	A.9.9
0x24 ... 0x26	Buffer manager access and keyhole registers	A.9.10
0x27	Not used	
0x28 ... 0x2F	Huffman decoder registers	A.9.13
0x30 ... 0x39	Inverse quantiser registers	A.9.14
0x3A ... 0x3B	Not used	
0x3C	Reserved	
0x3D ... 0x3F	Not used	
0x40 ... 0x7F	Test registers	

Table A.9.5 Overview of Spatial Decoder memory map

Addr. (hex)	Bit num.	Register Name	Page references
0x03	7	idct_too_few_mask	
	6	idct_too_many_mask	
	5	accept_enable_mask	
	4	target_met_mask	
	3	counter_flushed_too_early_mask	
	2	counter_flushed_mask	
	1	parser_mask	
	0	huffman_mask	

Table A.9.6 Interrupt service area registers (contd)

Addr. (hex)	Bit num.	Register Name	Page references
0x0B	7:0	Test register length_count	
0x0C	7:0		
0x0D	7:2	not used	
	1:0	start_code_detector_coding_standard	
0x0E	7:0	start_value	
0x0F	7:4	not used	
	3:0	picture_number	

Table A.9.7 Start code detector and input circuit registers (contd)

Addr. (hex)	Bit num.	Register Name	Page references
0x10	7:1	not used	
	0	startup_access *CED_BS_ACCESS*	
0x11	7:3	not used	
	2:0	bit_count_prescale *CED_BS_PRESCALE*	
0x12	7:0	bit_count_target *CED_BS_TARGET*	
0x13	7:0	bit_count *CED_BS_COUNT*	
0x14	7:1	not used	
	0	offchip_queue *CED_BS_QUEUE*	
0x15	7:1	not used	
	0	enable_stream *CED_BS_ENABLE_NXT_STM*	

Table A.9.8 Buffer start-up registers

Addr. (hex)	Bit num.	Register Name	Page references
0x1B	7:4	not used	
	3:0	refresh_cycle_length	
0x1C	7:4	not used	
	3:0	CAS_falling	
0x1D	7:4	not used	
	3:0	RAS_falling	
0x1E	7:1	not used	
	0	interface_timing_access	
0x1F	7:0	refresh_interval	
0x20	7	not used	
	6:4	DRAM_addr_strength[2:0]	
	3:1	CAS_strength[2:0]	
	0	RAS_strength[2]	
0x21	7:6	RAS_strength[1:0]	
	5:3	OEWE_strength[2:0]	
	2:0	DRAM_data_strength[2:0]	
0x22	7	ACCESS bit for pad strength etc. ?not used*CED_DRAM_CONFIGURE*	
	6	zero_buffers	
	5	DRAM_enable	
	4	no_refresh	
	3:2	row_address_bits[1:0]	
	1:0	DRAM_data_width[1:0]	
0x23	7:0	Test registers *CED_PLL_RES_CONFIG*	

Table A.9.9 DRAM interface configuration registers (contd)

Addr. (hex)	Bit num.	Register Name	Page references
0x24	7:1	not used	
	0	buffer_manager_access	
0x25	7:6	not used	
	5:0	buffer_manager_keyhole_address	
0x26	7:0	buffer_manager_keyhole_data	

Table A.9.10 Buffer manager access and keyhol registers

Addr. (hex)	Bit num.	Register Name	Page references
0x20	7:0	not used	
0x21	7:0	buffer_limit	
0x22	7:0		
0x23	7:0		
0x24	7:4	not used	
	3	cdb_full	
	2	cdb_empty	
	1	tb_full	
	0	tb_empty	

Table A.9.11 Buffer manager extended address space (contd)

Addr. (hex)	Bit num.	Register Name	Page references
0x28	7	demux_access *CED_H_CTRL[7]*	
	6:4	huffman_error_code[2:0] *CED_H_CTRL[6:4]*	
	3:0	private huffman control bits [3] selects special CBP, [2] selects 4/8 bit fixed length CBP	
0x29	7:0	parser_error_code *CED_H_DMUX_ERR*	
0x2A	7:4	not used	
	3:0	demux_keyhole_address *CED_H_KEYHOLE_ADDR*	
0x2B	7:0		
0x2C	7:0	demux _keyhole_data *CED_H_KEYHOLE*	
0x2D	7	dummy_last_picture *CED_H_ALU_REG0, r_dummy_last_frame_bit*	
	6	field_info *CED_H_ALU_REG0, r_field_info_bit*	
	5:1	not used	
	0	continue *CED_H_ALU_REG0, r_continue_bit*	
0x2E	7:0	rom_revision *CED_H_ALU_REG1*	
0x2F	7:0	private register	

Table A.9.12 Video demux registers

Addr. (hex)	Bit num.	Register Name	Page references
0x00 0x0F	7:0	not used	
0x10	7:0	horiz_pels *r_horiz_pels*	
0x11	7:0		
0x12	7:0	vert_pels *r_vert_pels*	
0x13	7:0		
0x14	7:2	not used	
	1:0	buffer_size *r_buffer_size*	
0x15	7:0		
0x16	7:4	not used	
	3:0	pel_aspect *r_pel_aspect*	
0x17	7:2	not used	
	1:0	bit_rate *r_bit_rate*	
0x18	7:0		
0x19	7:0		
0x1A	7:4	not used	
	3:0	pic_rate *r_pic_rate*	
0x1B	7:1	not used	
	0	constrained *r_constrained*	
0x1C	7:0	picture_type	
0x1D	7:0	h261_pic_type	

Table A.9.13 Video demux extended address space (Sheet 1 of 8)

Addr. (hex)	Bit num.	Register Name	Page references
0x33	7:2	not used	
	1:0	dc_huff_3	
0x34	7:2	not used	
	1:0	ac_huff_0	
0x35	7:2	not used	
	1:0	ac_huff_1	
0x36	7:2	not used	
	1:0	ac_huff_2	
0x37	7:2	not used	
	1:0	ac_huff_3	
0x38	7:2	not used	
	1:0	tq_0 *r_tq_0*	
0x39	7:2	not used	
	1:0	tq_1 *r_tq_1*	
0x3A	7:2	not used	
	1:0	tq_2 *r_tq_2*	
0x3B	7:2	not used	
	1:0	tq_3 *r_tq_3*	
0x3C	7:0	component_name_0 *r_c_0*	
0x3D	7:0	component_name_1 *r_c_1*	
0x3E	7:0	component_name_2 *r_c_2*	
0x3F	7:0	component_name_3 *r_c_3*	
0x40 0x63	7:0	private registers	
0x40	7:0	r_dc_pred_0	
0x41	7:0		
0x42	7:0	r_dc_pred_1	
0x43	7:0		
0x44	7:0	r_dc_pred_2	
0x45	7:0		
0x46	7:0	r_dc_pred_3	
0x47	7:0		
0x48 0x4F	7:0	not used	

Table A.9.13 Video demux ext nded address space (Sheet 3 of 8)

Addr. (hex)	Bit num.	Register Name	Page references
0x79	7:2	not used	
	1:0	blocks_h_1 r_blk_h_1	
0x7A	7:2	not used	
	1:0	blocks_h_2 r_blk_h_2	
0x7B	7:2	not used	
	1:0	blocks_h_3 r_blk_h_3	
0x7C	7:2	not used	
	1:0	blocks_v_0 r_blk_v_0	
0x7D	7:2	not used	
	1:0	blocks_v_1 r_blk_v_1	
0x7E	7:2	not used	
	1:0	blocks_v_2 r_blk_v_2	
0x7F	7:2	not used	
	1:0	blocks_v_3 r_blk_v_3	
0x7F 0xFF	7:0	not used	
0x100 0x10F	7:0	dc_bits_0[15:0] CED_H_KEY_DC_CPB0	
0x110 0x11F	7:0	dc_bits_1[15:0] CED_H_KEY_DC_CPB1	
0x120 0x13F	7:0	not used	
0x140 0x14F	7:0	ac_bits_0[15:0] CED_H_KEY_AC_CPB0	
0x150 0x15F	7:0	ac_bits_1[15:0] CED_H_KEY_AC_CPB1	
0x160 0x17F	7:0	not used	
0x180	7:0	dc_zssss_0 CED_H_KEY_ZSSSS_INDEX0	
0x181	7:0	dc_zssss_1 CED_H_KEY_ZSSSS_INDEX1	
0x182 0x187	7:0	not used	
0x188	7:0	ac_eob_0 CED_H_KEY_EOB_INDEX0	

Table A.9.13 Video demux extended address space (Sheet 5 of 8)

Addr. (Hex)	Bit num.	Register Name	Page references
0x850 0x85F	7:0	CED_KEY_MTYPE_P_CPB	
0x860 0x86F	7:0	CED_KEY_MTYPE_B_CPB	
0x870 0x88F	7:0	CED_KEY_MTYPE_H.261_CPB	
0x880 0x900	7:0	not used	
0x901	7:0	CED_KEY_HDSTROM_0	
0x902	7:0	CED_KEY_HDSTROM_1	
0x903 0x90F	7:0	CED_KEY_HDSTROM_2	
0x910 0xAB F	7:0	not used	
0xAC 0	7:0	CED_KEY_DMX_WORD_0	
0xAC 1	7:0	CED_KEY_DMX_WORD_1	
0xAC 2	7:0	CED_KEY_DMX_WORD_2	
0xAC 3	7:0	CED_KEY_DMX_WORD_3	
0xAC 4	7:0	CED_KEY_DMX_WORD_4	
0xAC 5	7:0	CED_KEY_DMX_WORD_5	
0xAC 6	7:0	CED_KEY_DMX_WORD_6	
0xAC 7	7:0	CED_KEY_DMX_WORD_7	

Table A.9.13 Video demux extended address space (Sheet 7 of 8)

Addr. (hex)	Bit num.	Register Name	Page references
0x36	7:2	not used	
	1:0	test register mpeg_indirection	
0x37	7:0	not used	
0x38	7:0	iq_table_keyhole_address	
0x39	7:0	iq_table_keyhole_data	

Table A.9.14 Inverse quantizer registers (contd)

Addr. (hex)	Register Name	Page references
0x00:0x3F	JPEG inverse quantisation table 0 MPEG default intra table	
0x40:0x7F	JPEG inverse quantisation table 1 MPEG default non-intra table	
0x80:0xBF	JPEG inverse quantisation table 2 MPEG down-loaded intra table	
0xC0:0xFF	JPEG inverse quantisation table 3 MPEG down-loaded non-intra table	

Table A.9.15 Iq table extended address space

A.10.1 Th cod d data port

Signal Name	Input / Output	Description
coded_clock	Input	A clock operating at up to 30 MHz controlling the operation of the input circuit.
coded_data[7:0]	Input	The standard 11 wires required to implement a Token Port transferring 8 bit data values. See section A.4 for an electrical description of this interface. Circuits off-chip must package the coded data into Tokens.
coded_extn	Input	
coded_valid	Input	
coded_accept	Output	
byte_mode	Input	When high this signal indicates that information is to be transferred across the coded data port in *byte mode* rather than *Token mode*.

Table A.10.1 Coded data port signals

(See Table A.9.7).

When configured for Token input via the MPI, the current Token is extended with the current value of coded_extn each time a value is written into coded_data[7:0]. Software is responsible for setting coded_extn to 0 before the last word of any Token is written to coded_data[7:0].

For example, a DATA Token is started by writing 1 into coded_extn and then 0x04 into coded_data[7:0]. The start of this new DATA Token then passes into the Spatial Decoder for processing.

Each time a new 8 bit value is written to coded_data[7:0], the current Token is extended. Coded_extn need only be accessed again when terminating the current Token, e.g. to introduce another Token. The last word of the current Token is indicated by writing 0 to coded_extn followed by writing the last word of the current Token into coded data[7:0].

Register name	Size/Dir.	Reset State	Description
coded_extn	1 rw	x	Tokens can be supplied to the Spatial Decoder via the MPI by writing to these registers.
coded_data[7:0]	8 w	x	
coded_busy	1 r	1	The state of this registers indicates if the Spatial Decoder is able to accept Tokens written into coded_data[7:0]. The value 1 indicates that the interface is busy and unable to accept data. Behaviour is undefined if the user tries to write to coded_data[7:0] when coded_busy = 1.
enable_mpi_input	1 rw	0	The value in this function enable registers controls whether coded data input to the Spatial Decoder is via the coded data port (0) or via the MPI (1).

Table A.10.2 Coded data input registers

Previous mode	Next Mode	Behaviour
Token	Byte	The off-chip circuitry supplying the Token in Token mode is responsible for completing the Token (i.e. with the extn bit of the last byte of information set to 0) before selecting byte mode.
	MPI input	Access to input via the MPI will not be granted (i.e. coded_busy will remain set to 1) until the off-chip circuitry supplying the Token in Token mode has completed the Token (i.e. with the extn bit of the last byte of information set to 0).
MPI input	Byte	The control software must have completed the Token (i.e. with the extn bit of the last byte of information set to 0) before enable_mpi_input is set to 0.
	MPI input	

Table A.10.3 Switching data input modes (contd)

The first byte supplied in byte mode causes a DATA Token header to be generated on-chip. Any further bytes transferred in byte mode are thereafter appended to this DATA Token until the input mode changes. Recall, DATA Tokens can contain as many bits as are necessary.

The MPI register bit, coded busy, and the signal, coded_accept, indicate on which interface the Spatial decoder is willing to accept data. Correct observation of these signals ensures that no data is lost.

A.10.4 Rate of accepting coded data

In the present invention, the input circuit passes Tokens to the Start Code Detector (see section A.11). The Start code Detector analyses data in the DATA Tokens bit serially. The Detector's normal rate of

SECTION A.11 Start code detector

A.11.1 Start codes

As is well known in the art, MPEG and H.261 coded video streams contain identifiable bit patterns called start codes. A similar function is served in JPEG by marker codes. Start/marker codes identify significant parts of the syntax of the coded data stream. The analysis of start/marker codes performed by the Start Code Detector is the first stage in parsing the coded data. The Start Code Detector is the first block on the Spatial Decoder following the input circuit.

The start/marker code patterns are designed so that they can be identified without decoding the entire bitstream. Thus, they can be used in accordance with the present invention, to help with error recovery and decoder start-up. The Start Code Detector provides facilities to detect errors in the coded data construction and to assist the start-up of the decoder.

A.11.2 Start code detector registers

As previously discussed, many of the Start Code Detector registers are in constant use by the Start Code Detector. So, accessing these registers will be unreliable if the Start Code Detector is processing data. The user is responsible for ensuring that the Start Code Detector is halted before accessing its registers.

The register start_code_detector_access is used to halt the Start Code Detector and so allow access to its registers. The Start Code Detector will halt after it generates an interrupt.

There are further constraints on when the start code search and discard all data modes can be initiated. These are described in A.11.8 and A.11.5.1.

Register name	Size/Dir.	Reset State	Description
illegal_length_count_event	1 rw	0	An illegal length count event will occur if while decoding JPEG data, a length count field is found carrying a value less than 2. This should only occur as the result of an error in the JPEG data.
illegal_length_count_mask	1 rw	0	If the mask register is set to 1 then an interrupt can be generated and the start code detector will stop. Behaviour following an error is not predictable if this error is suppressed (mask register set to 0). See A.11.4.1
jpeg_overlapping_start_event	1 rw	0	If the coding standard is JPEG and the sequence 0xFF 0xFF is found while looking for a marker code this event will occur.
jpeg_overlapping_start_mask	1 rw	0	This sequence is a legal stuffing sequence. If the mask register is set to 1 then an interrupt can be generated and the start code detector will stop. See A.11.4.2
overlapping_start_event	1 rw	0	If the coding standard is MPEG or H.261 and an overlapping start code is found while looking for a start code this event will occur.
overlapping_start_mask	1 rw	0	If the mask register is set to 1 then an interrupt can be generated and the start code detector will stop. See A.11.4.2

Table A.11.1 Start code detector registers (Sheet 2 of 5)

Register name	Size/Dir.	Reset State	Description
non_aligned_start_event	1 rw	0	When ignore_non_aligned is set to 1, start codes that are not byte aligned are ignored (treated as normal data). When ignore_non_aligned is set to 0, H.261 and MPEG start codes will be detected regardless of byte alignment and the non-aligned start event will be generated. If the mask register is set to 1 then the event will cause an interrupt and the start code detector will stop. See A.11.6 If the coding standard is configured as JPEG ignore_non_aligned is ignored and the non-aligned start event will never be generated.
non_aligned_start_mask	1 rw	0	
ignore_non_aligned	1 rw	0	
discard_extension_data	1 rw	1	When these registers are set to 1 extension or user data that cannot be decoded by the Spatial Decoder is discarded by the start code detector. See A.11.3.3
discard_user_data	1 rw	1	
discard_all_data	1 rw	0	When set to 1 all data and Tokens are discarded by the start code detector. This continues until a **FLUSH** Token is supplied or the register is set to 0 directly. The **FLUSH** Token that resets this register is discarded and not output by the start code detector. See A.11.5.1
insert_sequence_start	1 rw	1	See A.11.7

Table A.11.1 Start code detector registers (Sheet 4 of 5)

Register name	Size/Dir.	Reset State	Description
length_count	16 r0	0	This register contains the current value of the JPEG length count. This register is modified under the control of the coded data clock and should only be read via the MPI when the start code detector is stopped.

Table A.11.2 Start code detector test registers

A.11.3 Conversion of start codes to Tokens

In normal operation the function of the Start Code Detector is to identify start codes in the data stream and to then convert them to the appropriate start code Token. In the simplest case, data is supplied to the Start code Detector in a single long DATA Token. The output of the Start Code Detector is a number of shorter DATA Tokens interleaved with start code Tokens.

Alternatively, in accordance with the present invention, the input data to the Start Code Detector could be divided up into a number of shorter DATA Tokens. There is no restriction on how the coded data is divided into DATA Tokens other than that each DATA Token must contain 8 x n bits where n is an integer.

Other Tokens can be supplied directly to the input of the Start Code Detector. In this case, the Tokens are passed through the Start Code Detector with no processing

Start code Token generated	Start Code Value			
	MPEG (hex)	H.261 (hex)	JPEG (hex)	JPEG (name)
PICTURE_START	0x00	0x00	0xDA	SOS
SLICE_START [a]	0x01 to 0xAF	0x01 to 0xCC	0xD0 to 0xD7	RST_0 to RST_7
SEQUENCE_START	0xB3		0xD8	SOI
SEQUENCE_END	0xB7		0xD9	EOI
GROUP_START	0xB8		0xC0	SOF_0[b]
USER_DATA	0xB2		0xE0 to 0xEF	APP_0 to APP_F
			0xFE	COM
EXTENSION_DATA	0xB5		0xC8	JPG
			0xF0 to 0xFD	JPG_0 to JPG_D
			0xC2 to 0xBF	RES
			0xC1 to 0xCB	SOF_1 to SOF_{11}
			0xCC	DAC
DHT_MARKER			0xC4	DHT
DNL_MARKER			0xDC	DNL
DQT_MARKER			0xDB	DQT
DRI_MARKER			0xDD	DRI

Table A.11.4 Tokens from start code values

a. This Token contains an 8 bit data field which is loaded with a value determined by the start code value.

b. Indicates start of baseline DCT encoded data.

A.11.3.4 JPEG Table definitions

JPEG supports down loaded Huffman and quantizer tables. In JPEG data, the definition of these tables is preceded by the marker codes DNL and DQT. The Start Code Detector generates the Tokens DHT_MARKER and DQT_MARKER when these marker codes are detected. These Tokens indicate to the Video Demux that the DATA Token which follows contains coded data describing Huffman or quantizer table (using the formats described in JPEG).

A.11.4 Error detection

The Start Code Detector can detect certain errors in the coded data and provides some facilities to allow the decoder to recover after an error is detected (see A.11.8, "Start code searching").

A.11.4.1 Illegal JPEG length count

Most JPEG marker codes have a 16 bit length count field associated with them. This field indicates how much data is associated with this marker code. Length counts of 0 and 1 are illegal. An illegal length should only occur following a data error. In the present invention, this will generate an interrupt if illegal_length_count_mask is set to 1.

Recovery from errors in JPEG data is likely to require additional application specific data due to the difficulty of searching for start codes in JPEG data (see A.11.8.1).

A.11.4.2 Overlapping start/marker codes

In the present invention, overlapping start codes should only occur following a data error. An MPEG, byte aligned, overlapping start code is illustrated in Figure 64. Here, the Start Code Detector first sees a pattern that looks like a picture start code. Next the Start Code Detector sees that this picture start code is overlapped with a group start. Accordingly, the Start Code Detector to correct.

A.11.4.4 Sequence of event g n ration

In the present invention, certain coded data patterns (probably indicating an error condition) will cause more than one of the above error conditions to occur within a short space of time. Consequently, the sequence in which the Start Code Detector examines the coded data for error conditions is:

1) Non-aligned start codes
2) Overlapping start codes
3) Unrecognized start codes

Thus, if a non-aligned start code overlaps another, later, start code, the first event generated will be associated with the non-aligned start code. After this event has been serviced, the Start Code Detector's operation will proceed, detecting the overlapped start code a short time later.

The Start Code Detector only attempts to recognize the start code after all tests for non-aligned and overlapping start codes are complete.

A.11.5 Decoder start-up and shutdown

The Start Code Detector provides facilities to allow the current decoding task to be completed cleanly and for a new task to be started.

There are limitations on using these techniques with JPEG coded video as data segments can contain values that emulate marker codes (see A.11.8.1).

A.11.5.1 Clean end to decoding

The Start Code Detector can be configured to generate an interrupt and stop once the data for the current picture is complete. This is done by setting stop_after_picture = 1 and stop_after_picture_mask = 1.

Once the end of a picture passes through the Start Code Detector, a FLUSH Token is generated (A.11.7.2), from ~~one~~ part of one coded video sequence to another. In this example, the filing system only allows access to "blocks" of data. This block structure might be derived from the sector size of a disc or a block error correction system. So, the position of entry and exit points in the coded video data may not be related to the filing system block structure.

The stop_after_picture and discard_all_data mechanisms allow unwanted data from the old video sequence to be discarded. Inserting a FLUSH Token after the end of the last filing system data block resets the discard_all_data mode. The start code search mode can then be used to discard any data in the next data block that precedes a suitable entry point.

A.11.6 Byte alignment

As is well known in the art, the different coding schemes have quite different views about byte alignment of start/marker codes in the data stream.

For example, H.261 views communications as being bit serial. Thus, there is no concept of byte alignment of start codes. By setting ignore_non_aligned = 0 the Start Code Detector is able to detect start codes with any bit alignment. By setting non-aligned_start_mask = 0, the start code non-alignment interrupt is suppressed.

In contrast, however, JPEG was designed for a computer environment where byte alignment is guaranteed. Therefore, marker codes should only be detected when byte aligned. When the coding standard is configured as JPEG, the register ignore_non_aligned is ignored and the non-aligned start event will never be generated. However, setting ignore_non_aligned = 1 and non_aligned_start_mask = 0 is recommended to ensure compatibility with future products.

MPEG, on the other hand, was designed to meet the needs of both communications (bit serial) and computer (byte

A.11.7 Aut matic Token generation

In the present invention, most of the Tokens output by the Start Code Detector directly reflect syntactic elements of the various picture and video coding standards. In addition to these "natural" Tokens,some useful "invented" Tokens are generated. Examples of these proprietary tokens are PICTURE_END and CODING_STANDARD. Tokens are also introduced to remove some of the syntactic differences between the coding standards and to "tidy up" under error conditions.

This automatic Token generation is done after the serial analysis of the coded data (see Figure 61, "The Start Code Detector"). Therefore the system responds equally to Tokens that have been supplied directly to the input of the Spatial Decoder via the Start Code Detector and to Tokens that have been generated by the Start Code Detector following the detection of start codes in the coded data.

A.11.7.1 Indicating the end of a picture

In general, the coding standards don't explicitly signal the end of a picture. However, the Start Code Detector of the present invention generates a PICTURE_END Token when it detects information that indicates that the current picture has been completed.

The Tokens that cause PICTURE_END to be generated are: SEQUENCE_START, GROUP_START, PICTURE_START, SEQUENCE_END and FLUSH.

A.11.7.2 Stop after picture end option

If the register stop_after_picture is set, then the Start Code Detector will stop after a PICTURE_END Token has passed through. However, a FLUSH Token is inserted after the PICTURE_END to "push" the tail end of the coded data through the decoder and to reset the system. See A.11.5.1.

start_code_search	Start codes searched for ...
0 [a]	Normal operation
1	Reserved (will behave as discard data)
2	
3	sequence start
4	group or sequence start
5 [b]	picture, group or sequence start
6	slice, picture, group or sequence start
7	the next start or marker code

Table A.11.6 Start code search modes

a. A FLUSH Token places the Start Code Detector in this search mode.

b. This is the default mode after reset.

When a non-zero value is written into the start_code_search register, the Start Code Detector will start to discard all incoming data until the specified start code is detected. The start_code_search register will then reset to 0 and normal operation will continue. The start code search will start immediately after a non-zero value is written into the start_code_search register. The result will be unpredictable if this is done when the Start Code Detector is actively processing data. So, before initiating a start code search, the Start Code Detector should be stopped so no data is being processed. The Start Code Detector is always in this condition if any of the Start Code Detector events (non-aligned start event etc.) has just generated an interrupt.

A.11.8.1 Limitations on using start code search with JPEG

~~SECTION~~ A.12 Decoder start-up control

A.12.1 Overview of d coder start-up

In a decoder, video display will normally be delayed a short time after coded data is first available. During this delay, coded data accumulates in the buffers in the decoder. This pre-filling of the buffers ensures that the buffers never empty during decoding and, this, therefore ensures that the decoder is able to decode new pictures at regular intervals.

Generally, two facilities are required to correctly start-up a decoder. First, there must be a mechanism to measure how much data has been provided to the decoder. Second, there must be a mechanism to prevent the display of a new video stream. The Spatial Decoder of the invention provides a *bit counter* near its input to measure how much data has arrived and an *output gate* near its output to prevent the start of new video stream being output.

There are three levels of complexity for the control of these facilities:

- Output gate always open
- Basic control
- Advanced control

With the output gate always open, picture output will start as soon as possible after coded data starts to arrive at the decoder. This is appropriate for still picture decoding or where display is being delayed by some other mechanism.

The difference between basic and advanced control relates to how many short video streams can be accommodated in the decoder's buffers at any time. Basic control is sufficient for most applications. However, advanced control allows user software to help the decoder manage the start-up of several very short video streams.

A.12.4 Start-up control r gisters

Register name	Size/Dir.	Reset State	Description
startup_access *CED_BS_ACCESS*	1 rw	0	Writing 1 to this register requests that the bit counter and gate opening logic stop to allow access to their configuration registers.
bit_count *CED_BS_COUNT*	8 rw	0	This bit counter is incremented as coded data leaves the start code detector. The number of bits required to increment bit_count once is
bit_count_prescale *CED_BS_PRESCALE*	3 rw	0	approx. $2^{(\text{bit\_count\_prescale}+1)}$ x 512. The bit counter starts counting bits after a **FLUSH** Token passes through the bit counter It is reset to zero and then stops incrementing after the bit count target has been met.
bit_count_target *CED_BS_TARGET*	8 rw	x	This register specifies the bit count target. A target met event is generated whenever the following condition becomes true: **bit_count >= bit_count_target**
target_met_event *BS_TARGET_MET_EVENT*	1 rw	0	When the bit count target is met this event will be generated. If the mask register is set to 1
target_met_mask	1 rw	0	then an interrupt can be generated, however, the bit counter will NOT stop processing data. This event will occur when the bit counter increments to its target. It will also occur if a target value is written which is less than or equal to the current value of the bit counter Writing 0 to bit_count_target will always generate a target met event.

Table A.12.1 Decoder start-up registers

Register name	Size/Dir.	Reset State	Description
accept_enable_event *SS_STREAM_END_EVENT*	1 rw	0	This event indicates that a FLUSH Token has passed through the output gate (causing it to close) and that an enable was available to allow the gate to open. If the mask register is set to 1 then an interrupt can be generated and the register **enable_stream** will be reset. See A.12.7.1
accept_enable_mask	1 rw	0	

Table A.12.1 Decoder start-up registers (contd)

vbv_delay for the first picture in each new stream

Counter flushed too early service to react to this condition

The video demux (also known as the video parser) can generate an interrupt when it decodes the vbv_delay for a new video stream (i.e., the first picture to arrive at the video demux after a FLUSH). The interrupt service routine should compute an appropriate value for bit_count_target and write it. When the bit counter reaches this target, it will insert an enable into a short queue between the bit counter and the output gate. When the output gate opens it removes an enable from this queue.

output gate is open . Streams B and C have met their start-up conditions and are entirely contained within the buffers managed by the Spatial Decoder. Stream D is still arriving at the input of the Spatial Decoder.

Enables for streams B and C are in the queue. So, when stream A is completed B will be able to start immediately. Similarly C can follow immediately behind B.

If A is still passing through the output gate when D meets its start-up target an enable will be added to the queue, filling the queue. If no enables have been removed from the queue by the time the end of D passes the bit counter (i.e., A is still passing through the output gate) no new stream will be able to start through the bit counter. Therefore, coded data will be held up at the input until A completes and an enable is removed from the queue as the output gate is opened to allow B to pass through.

A.12.7 Advanced operation

In accordance with the present invention, advanced control of the start-up logic allows user software to infinitely extend the length of the enable queue described in A.12.6, "Basic operation". This level of control will only be required where the video decoder must accommodate a series of short video streams longer than that described in A.12.6.2, "A succession of short streams".

In addition to the configuration required for Basic operation of the system, the following configurations are required after reset (having gained access to the start-up control logic by writing 1 to start_up access):

- set offchip_queue = 1
- set accept_enable_mask = 1 to enable interrupts when an enable has been removed from the queue
- set target_met_mask = 1 to enable interrupts when a stream's bit count target is met

required to increment the bit counter once. Furthermore, bit_count_prescale is a 3 bit register than can hold a value between 0 and 7.

n	Range (bits)	Resolution (bits)
0	0 to 262144	1024
1	0 to 524288	2048
7	0 to 31457280	122880

Table A.12.2 Example bit counter ranges

The bit count is approximate, as some elements of the video stream will already have been Tokenized (e.g., the start codes) and, therefore includes non-data Tokens.

A.12.10 Counter flushed too early

If a FLUSH token arrives at the bit counter before the bit count target is attained, an event is generated which can cause an interrupt (if counter_flushed_too_early_mask = 1). If the interrupt is generated, then the bit counter circuit will stop, preventing further data input. It is the responsibility of the user's software to decide when to open the output gate after this event has occurred. The output gate can be made to open by writing 0 as the bit count target. These circumstances should only arise when trying to decode video streams that last only a few pictures.

consideration when polling such registers as cdb_full and cdb_empty to monitor buffer conditions.

Register name	Size/Dir.	Reset State	Description
buffer_manager_access	1 rw	1	This access bit stops the operation of the buffer manager so that its various registers can be accessed reliably. See A.6.4.1 Note: this access register is unusual as its default state after reset is 1. i.e. after reset the buffer manager is halted awaiting configuration via the microprocessor interface.

Register name	Size/Dir.	Reset State	Description
buffer_manager_keyhole_address	6 rw	x	Keyhole access to the extended access space used for the buffer manager registers shown below. See A.6.4.3 for more information about accessing registers through a keyhole.
buffer_manager_keyhole_data	8 rw	x	
buffer_limit	18 rw	x	This specifies the overall size of the DRAM array attached to the Spatial Decoder. All buffer addresses are calculated MOD this buffer size and so will wrap round within the DRAM provided.
cdb_base	18 rw	x	These registers point to the base of the coded data (cdb) and Token (tb) buffers.
tb_base			
cdb_length	18 rw	x	These registers specify the length (i.e. size) of the coded data (cdb) and Token (tb) buffers.
tb_length			
cdb_read	18 ro	x	These registers hold an offset from the buffer base and indicate where data will be read from next.
tb_read			
cdb_number	18 ro	x	These registers show how much data is currently held in the buffers.
tb_number			
cdb_full	1 ro	x	These registers will be set to 1 if the coded data (cdb) or Token (tb) buffer fills
tb_full			
cdb_empty	1 ro	x	These registers will be set to 1 if the coded data (cdb) or Token (tb) buffer empties.
tb_empty			

Table A.13.1 Buffer manager registers (contd)

not h~~ave~~ a "real-time" requirement will not need the large off-chip buffers supported by the buffer manager. In this case, the DRAM interface can be configured (by writing 1 to the zero_buffers register) to ignore the buffer manager to provide a 128 bit stream on-chip FIFO for the coded data buffer and the Token buffers.

The zero buffers option may also be appropriate for applications which operate working at low data rates and with small picture formats.

Note: the zero_buffers register is part of the DRAM interface and, therefore, should be set only during the post-reset configuration of the DRAM interface.

A.13.4 Buffer operation

The data transfer through the buffers is controlled by a handshake Protocol. Hence, it is guaranteed that no data errors will occur if the buffer fills or empties. If a buffer is filled, then the circuits trying to send data to the buffer will be halted until there is space in the buffer. If a buffer continues to be full, more processing stages "up steam" of the buffer will halt until the Spatial Decoder is unable to accept data on its input port. Similarly, if a buffer empties, then the circuits trying to remove data from the buffer will halt until data is available.

As described in A.13.2, the position and size of the coded data and Token buffer are specified by the buffer base and length registers. The user is responsible for configuring these registers and for ensuring that there is no conflict in memory usage between the two buffers.

Register name	Size/Dir.	Reset State	Description
dummy_last_picture *CED_H_ALU_REG0* *r_rom_control* *r_dummy_last_frame_bit*	1 rw	0	When this register is set to 1 the Video Demux will generate information for a "dummy" Intra picture as the last picture of an MPEG sequence. This function is useful when the Temporal Decoder is configured for automatic picture re-ordering (see A.18.3.5, "Picture sequence re-ordering", to flush the last P or I picture out of the Temporal Decoder. No "dummy" picture is required if: • the Temporal Decoder is not configured for re-ordering • another MPEG sequence will be decoded immediately (as this will also flush out the last picture) • the coding standard is not MPEG
field_info *CED_H_ALU_REG0* *r_rom_control* *r_field_info_bit*	1 rw	0	When this register is set to 1 the first byte of any MPEG extra_information_picture is placed in the FIELD_INFO Token. See A.14.7.1
continue *CED_H_ALU_REG0* *r_rom_control* *r_continue_bit*	1 rw	0	This register allows user software to control how much extra, user or extension data it wants to receive when is it is detected by the decoder. See A.14.6 and A.14.7
rom_revision *CED_H_ALU_REG1* *r_rom_revision*	8 ro		Immediately following reset this holds a copy of the microcode ROM revision number. This register is also used to present to control software data values read from the coded data. See A.14.6, "Receiving User and Extension data", and A.14.7, "Receiving Extra Information".

Table A.14.1 Top level Video Demux registers (contd)

Register name	Size/Dir.	Reset State	Description
max_h	2 rw	x	These registers hold the macroblock width and height in blocks (8 x 8 pixels). The values 0 to 3 indicate a width/height of 1 to 4 blocks. See section A.14.2
max_v	2 rw	x	
max_component_id	2 rw	x	The values 0 to 3 indicate that 1 to 4 different video components are currently being decoded. See section A.14.2
Nf	8 rw	x	During JPEG operation this register holds the parameter Nf (number of image components in frame).
blocks_h_0 blocks_h_1 blocks_h_2 blocks_h_3	2 rw	x	For each of the 4 colour components the registers blocks_h_*n* and blocks_v_*n* hold the number of blocks horizontally and vertically in a macroblock for the colour component with component ID *n*. See section A.14.2
blocks_v_0 blocks_v_1 blocks_v_2 blocks_v_3	2 rw	x	
tq_0 tq_1 tq_2 tq_3	2 rw	x	The two bit value held by the register tq_*n* describes which Inverse Quantisation table is to be used when decoding data with component ID *n*.

Table A.14.2 Video demux picture construction registers (contd)

Register name	Size/Dir.	Reset State	Description
dc_huff_0 dc_huff_1 dc_huff_2 dc_huff_3	2 rw		The two bit value held by the register dc_huff_*n* describes which Huffman decoding table is to be used when decoding the DC coefficients of data with component ID *n*. Similarly ac_huff_*n* describes the table to be used when decoding AC coefficients. Baseline JPEG requires up to two Huffman tables per scan. The only tables implemented are 0 and 1.
ac_huff_0 ac_huff_1 ac_huff_2 ac_huff_3	2 rw		
dc_bits_0[15:0] dc_bits_1[15:0]	8 rw		Each of these is a table of 16, eight bit values. They provide the BITS information (see JPEG Huffman table specification) which form part of the description of two DC and two AC Huffman tables. See section A.14.3.1
ac_bits_0[15:0] ac_bits_1[15:0]	8 rw		
dc_huffval_0[11:0] dc_huffval_1[11:0]	8 rw		Each of these is a table of 12, eight bit values. They provide the HUFFVAL information (see JPEG Huffman table specification) which form part of the description of two DC Huffman tables. See section A.14.3.1
ac_huffval_0[161:0] ac_huffval_1[161:0]	8 rw		Each of these is a table of 162, eight bit values. They provide the HUFFVAL information (see JPEG Huffman table specification) which form part of the description of two AC Huffman tables. See section A.14.3.1
dc_zssss_0 dc_zssss_1	8 rw		These 8 bit registers hold values that are "special cased" to accelerate the decoding of certain frequently used JPEG VLCs. dc_ssss - magnitude of DC coefficient is 0 ac_eob - end of block ac_zrl - run of 16 zeros
ac_eob_0 ac_eob_1	8 rw		
ac_zrl_0 ac_zrl_1	8 rw		

Table A.14.3 Video demux Huffman table registers

Register name	Size/Dir.	Reset State	Description
picture_type	2 rw		During MPEG operation this register holds the picture type of the picture being decoded .
h_261_pic_type	8 rw		This register is loaded when decoding H.261 data. It holds information about the picture format. 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 r | r | s | d | f | q | r | r Flags: s - Split Screen Indicator d - Document Camera f - Freeze Picture Release This value is not used by the decoder chips. However, the information should be used when configuring **horiz_pels**, **vert_pels** and the display or output device.
broken_closed	2 rw		During MPEG operation this register holds the broken_link and closed_gop information for the group of pictures being decoded. 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 r | r | r | r | r | r | c | b Flags: c - closed_gop

Table A.14.4 Other Video Demux registers (contd)

Register name	Size/Dir.	Reset State	Description
restart_interval	8 rw		This register is loaded when decoding JPEG data with a value indicating the minimum start-up delay before decoding should start. See the MPEG standard for a definition of this value.

Table A.14.4 Other Video Demux registers (contd)

register	Token	standard	comment
component_name_*n*	COMPONENT_NAME	JPEG	in coded data.
		MPEG	not used in standard.
		H.261	
horiz_pels vert_pels	HORIZONTAL_SIZE VERTICAL_SIZE	MPEG	in coded data.
		JPEG	
		H.261	automatically derived from picture type.
horiz_macroblocks vert_macroblocks	HORIZONTAL_MBS VERTICAL_MBS	MPEG	control software must derive from horizontal and vertical picture size.
		JPEG	
		H.261	automatically derived from picture type.
max_h max_v	DEFINE_MAX_SAMPLING	MPEG	control software must configure. Sampling structure is fixed by standard.
		JPEG	in coded data.
		H.261	automatically configured for 4:2:0 video.

Table A.14.5 Register to Token cross reference

register	Token	standard	comment
dc_bits_0[15:0] dc_bits_1[15:0]	in DATA Token following DHT_MARKER Token	JPEG	in coded data.
dc_huffval_0[11:0] dc_huffval_1[11:0] dc_zssss_0 dc_zssss_1		MPEG	control software must configure.
		H.261	not used in standard.
ac_bits_0[15:0] ac_bits_1[15:0]	in DATA Token following DHT_MARKER Token	JPEG	in coded data.
ac_huffval_0[161:0] ac_huffval_1[161:0] ac_eob_0 ac_eob_1 ac_zrl_0 ac_zrl_1		MPEG	not used in standard.
		H.261	
buffer_size	VBV_BUFFER_SIZE	MPEG	in coded data.
		JPEG	not used in standard
		H.261	
pel_aspect	PEL_ASPECT	MPEG	in coded data.
		JPEG	not used in standard
		H.261	
bit_rate	BIT_RATE	MPEG	in coded data.
		JPEG	not used in standard
		H.261	
pic_rate	PICTURE_RATE	MPEG	in coded data.
		JPEG	not used in standard
		H.261	
constrained	CONSTRAINED	MPEG	in coded data.
		JPEG	not used in standard
		H.261	
picture_type	PICTURE_TYPE	MPEG	in coded data.
		JPEG	n t used in standard
		H.261	

Table A.14.5 Regist r to T ken cross reference (contd)

dimensions. The Spatial Decoder registers associated with this information are: horiz_pels, vert_pels, horiz_macroblocks and vert_macroblocks.

The Spatial Decoder registers, blocks_h_*n*, blocks_v_*n*, max_h, max_v and max_component_id specify the composition of the macroblocks (minimum coding units in JPEG). Each is a 2 bit register than can hold values in the range 0 to 3. All except max_component_id specify a block count of 1 to 4. For example, if register max_h holds 1, then a macroblock is two blocks wide. Similarly, max_component_id specifies the number of different color components involved.

	2:1:1	4:2:2	4:2:0	1:1:1
max_h	1	1	1	0
max_v	0	1	1	0
max_component_id	2	2	2	2
blocks_h_0	1	1	1	0
blocks_h_1	0	0	0	0
blocks_h_2	0	0	0	0
blocks_h_3	x	x	x	x
blocks_v_0	0	1	1	0
blocks_v_1	0	1	0	0
blocks_v_2	0	1	0	0
blocks_v_3	x	x	x	x

Table A.14.6 Configuration for various macroblock formats

DATA and DHT_MARKER Tokens to the input of the Spatial Decoder while the Spatial Decoder is configured for JPEG operation. This mechanism can be used for configuring the DC coefficient Huffman tables required for MPEG operation, however, the coding standard of the Spatial Decoder must be set to JPEG while the tables are down loaded.

E	7	6	5	4	3	2	1	0	Token Name
1	0	0	0	1	0	1	0	1	CODING_STANDARD
0	0	0	0	0	0	0	0	1	1 = JPEG
0	0	0	0	1	1	1	0	0	DHT_MARKER
1	0	0	0	0	0	1	x	x	DATA
1	t	t	t	t	t	t	t	t	T_h - Value indicating which Huffman table is to be loaded. JPEG allows 4 tables to be downloaded. Values 0x00 and 0x01 specify DC coefficient coding tables 0 and 1. Values 0x10 and 0x11 specifies AC coefficient coding tables 0 and 1.
1	n	n	n	n	n	n	n	n	L_i - 16 words carrying BITS information
:									
1	n	n	n	n	n	n	n	n	
1	n	n	n	n	n	n	n	n	V_{ij} - Words carrying HUFFVAL information (the number of words depends on the number of different symbols).
:									
e	n	n	n	n	n	n	n	n	e - the extension bit will be 0 if this is the endof the DATA Token or 1 if another table description is contained in the same DATA Token.

This sequence can be repeated to allow several tables to be described in a single Token

Table A.14.7 Huffman table configuration via Tokens

and this ensures that other decoder chips (such as the Temporal Decoder) are correctly configured.

A.14.4.3 MPEG Huffman tables

The majority of the Huffman coding tables required to decode MPEG are held in ROMs within the Spatial Decoder (again, in the parser state machine) and, thus, require no user intervention. The exceptions are the tables required for decoding the DC coefficients of Intral macroblocks. Two tables are required, one for chroma the other for luma. These must be configured by user software before decoding begins.

macroblock construction	CIF / QCIF	picture construction	CIF	QCIF
max_h	1	horiz_pels	352	176
max_v	1	vert_pels	288	144
max_component_id	2	horiz_macroblocks	22	11
blocks_h_0	1	vert_macroblocks	18	9
blocks_h_1	0			
blocks_h_2	0			
blocks_v_0	1			
blocks_v_1	0			
blocks_v_2	0			

Table A.14.8 Automatic settings for H.261

Table A.14.10 shows the sequence of Tokens required to configure the DC coefficient Huffman tables within the Spatial Decoder. Alternatively, the same results can be obtained by writing this information to registers via the MPI.

The registers dc_huff_*n* control which DC coefficient Huffman tables are used with each color component. Table

E	[7:0]	Token Name
1	0x00	**16 words carrying BITS information describing a total of 9 different VLCs:** 2, 2 bit codes; 3, 3 bit codes; 1, 4 bit codes; 1, 5 bit codes; 1, 6 bit codes; 1, 7 bit codes. If configuring via the MPI rather than with Tokens these values would be written into the **dc_bits_0[15:0]** registers.
1	0x02	
1	0x03	
1	0x01	
1	0x01	
1	0x01	
1	0x01	
1	0x00	
1	0x00	
1	0x00	
1	0x00	
1	0x00	
1	0x00	
1	0x00	
1	0x00	
1	0x00	
1	0x01	**9 words carrying HUFFVAL information** If configuring via the MPI rather than with Tokens these values would be written into the **dc_huffval_0[11:0]** registers.
1	0x02	
1	0x00	
1	0x03	
1	0x04	
1	0x05	
1	0x06	
1	0x07	
0	0x08	

Table A.14.10 MPEG DC Huffman table configuration (contd)

E	[7:0]	Token Name
1	0x15	CODING_STANDARD
0	0x02	2 = JPEG

Table A.14.10 MPEG DC Huffman table configuration (contd)

A.14.4.4 MPEG Picture structure

The macroblock construction *defined* for MPEG is the same as that used by H.261. The picture dimensions are encoded in the coded data.

For standard 4:2:0 operation, the macroblock characteristics should be configured as indicated in Table A.14.8. This can be done either by writing to the registers as indicated or by applying the equivalent Tokens (see Table A.14.5) to the input of the Spatial Decoder.

The approach taken to configure picture dimensions will depend upon the application. If the picture format is known before decoding starts, then the picture construction registers listed in Table A.14.8 can be initialized with appropriate values. Alternatively, the picture dimensions can be decoded from the coded data and used to configure the Spatial Decoder. In this case the user must service the parser error ERR_MPEG_SEQUENCE, see A.14.8, "Changes at the MPEG sequence layer".

subsequent scans.

To improve the performance of the Huffman decoding, certain commonly used symbols are specially cased. These are: DC coefficient with magnitude 0, end of block AC coefficients and run of 16 zero AC coefficients. The values for these special cases should be written into the appropriate registers.

A.14.4.6.1 Table selection

The registers dc_huff_*n* and ac_huff_*n* control which AC and DC coefficient Huffman tables are used with which color component. During JPEG operation, these relationships are defined by the TD_j and Ta_j fields of the scan header syntax.

A.14.4.7 JPEG Picture structure

There are two distinct levels of baseline JPEG decoding supported by the Spatial Decoder: up to 4 components per frame ($N_f \leq 4$) and greater than 4 components per frame ($N_f > 4$). If $N_f > 4$ is used, the control software required becomes more complex.

A.14.4.7.1 Nf≤4

The frame component specification parameters contained in the JPEG frame header configure the macroblock construction registers (see Table A.14.8) when they are decoded. No user intervention is required, as all the specifications required to decode the 4 different color components as defined.

For further details of the options provided by JPEG the reader should study the JPEG specification. Also, there is a short description of JPEG picture formats in § A.16.1.

A.14.4.7.2 JPEG with more than 4 components

The Spatial Decoder can decode JPEG files containing up to 256 different color components (the maximum permitted by JPEG). However, additional user intervention is required if more than 4 color component are to be decoded. JPEG only allows a maximum of 4 components in any scan.

indicated in parser_event. Thereafter, parser_mask determines whether an interrupt is generated. If parser_mask is set to 1, an interrupt will be generated and the Parser will halt. The register parser_error_code[7:0] will hold a value indicating the cause of event.

If 1 is written to huffman_event after servicing the interrupt, the Huffman decoder will attempt to recover from the error. Also, if huffman_mask was set to 0 (masking the interrupt and not halting the Huffman decoder) the Huffman decoder will attempt to recover form the error automatically.

If 1 is written to parser_event after servicing the interrupt, the Parser will start operation again. If the event indicated a bitstream error, the Video Demux will attempt to recover from the error.

If parser_mask was set to 0, the Parser will set its event bit, but will not generate an interrupt or halt. It will continue operation and attempt to recover from the error automatically.

parser_ rror_code(2:0)	Description
0x21	ERR_GSPARE H.261 GSARE information has been detected see A.14.7
0x22	ERR_PTYPE The value of the H.261 picture type has changed. The register h_261_pic_type can be inspected to see what the new value is.
0x30	ERR_JPEG_FRAME
0x31	ERR_JPEG_FRAME_LAST
0x32	ERR_JPEG_SCAN Picture size or Ns changed
0x33	ERR_JPEG_SCAN_COMP Component Change !
0x34	ERR_DNL_MARKER
0x40	ERR_MPEG_SEQUENCE One of the parameters communicated in the MPEG sequence layer has changed. See A.14.8
0x41	ERR_EXTRA_PICTURE MPEG extra_information_picture has been detected see A.14.7
0x42	ERR_EXTRA_SLICE MPEG extra_information_slice has been detected see A.14.7
0x43	ERR_VBV_DELAY The VBV_DELAY parameter for the first picture in a *new* MPEG video sequence has been detected by the Video Demux. The new value of delay is available in the register vbv_delay. The first picture of a new sequence is defined as the first picture after a sequence end. FLUSH or reset.
0x80	ERR_SHORT_TOKEN An incorrectly formed Token has been detected. This error should not occur during normal operation.

Table A.14.12 Parser err r cod s (Sheet 2 of 5)

parser_error_code[7:0]	Description
0xA6	ERR_RESTART_SKIP During JPEG operation a restart marker has been encountered either in in an unexpected place or the value of the restart marker is unexpected. If a restart marker is not found when one is expected the Huffman event "Found serial data when Token expected" will be generated.
0xB0	ERR_SKIP_INTRA During MPEG operation, a macro block with a macro block address increment greater than 1 has been found within an intra (I) picture. This is illegal and probably indicates a bitstream error.
0xB1	ERR_SKIP_DINTRA During MPEG operation, a macro block with a macro block address increment greater than 1 has been found within an DC only (D) picture. This is illegal and probably indicates a bitstream error.
0xB2	ERR_BAD_MARKER During MPEG operation, a marker bit did not have the expected value. This is probably indicates a bitstream error.
0xB3	ERR_D_MBTYPE During MPEG operation, within a DC only (D) picture, a macroblock was found with a macroblock type other than 1. This is illegal and probably indicates a bitstream error.
0xB4	ERR_D_MBEND During MPEG operation, within a DC only (D) picture, a macroblock was found with 0 in it's end of macroblock bit. This is illegal and probably indicates a bitstream error.
0xB5	ERR_SVP_BACKUP During MPEG operation, a slice has been encountered with a slice vertical position less than that expected. This is likely to indicate an error in the coded data
0xB6	ERR_SVP_SKIP_ROWS During MPEG operation, a slice has been encountered with a slice vertical position greater than that expected. This is likely to indicate an error in the coded data.
0xB7	ERR_FST_MBA_BACKUP During MPEG operation, a macroblock has been encountered with a macro block address less than that expected. This is likely to indicate an error in the coded data.

Table A.14.12 Parser error codes (Sheet 4 f 5)

T ken Name	MPEG	JPEG	H.261
ERR_JPEG_SCAN_COMP		✓	
ERR_DNL_MARKER		✓	
ERR_MPEG_SEQUENCE	✓		
ERR_EXTRA_PICTURE	✓		
ERR_EXTRA_SLICE	✓		
ERR_VBV_DELAY	✓		
ERR_SHORT_TOKEN	✓	✓	✓
ERR_H261_PIC_END_UNEXPECTED			✓
ERR_GN_BACKUP			✓
ERR_GN_SKIP_GOB			✓
ERR_NBASE_TAB		✓	
ERR_QUANT_PRECISION		✓	
ERR_SAMPLE_PRECISION		✓	
ERR_NBASE_SCAN		✓	
ERR_UNEXPECTED_DNL		✓	
ERR_EOS_UNEXPECTED		✓	
ERR_RESTART_SKIP		✓	
ERR_SKIP_INTRA	✓		
ERR_SKIP_DINTRA	✓		
ERR_BAD_MARKER	✓		
ERR_D_MBTYPE	✓		
ERR_D_MBEND	✓		
ERR_SVP_BACKUP	✓		
ERR_SVP_SKIP_ROWS	✓		
ERR_FST_MBA_BACKUP	✓		
ERR_FST_MBA_SKIP	✓		
ERR_PICTURE_END_UNEXPECTED	✓		
ERR_TST_PROGRAM	✓	✓	✓
ERR_NO_PROGRAM	✓	✓	✓
ERR_TST_END	✓	✓	✓
ERR_UCODE_ADDR	✓	✓	✓
ERR_NOT_IMPLEMENTED	✓	✓	✓

Table A.14.13 Parser error codes and the different standards (contd)

NOTE:

1)The first byte of the extension/user data is always presented via the rom_revision register regardless of the state of continue.

2)There is no event indicating that the last byte of extension/user data has been read.

A.14.7 Receiving Extra Information

H.261 and MPEG allow information extending the coding standard to be embedded within pictures and groups of blocks (H.261) or slices (MPEG). The mechanism is different from that used for extension and user data (described in Section A.14.6). No start code precedes the data and, thus, it cannot be deleted by the Start Code Detector.

During H.261 operation, the Parser events ERR_PSPARE and ERR_GSPARE indicate the detection of this information. The corresponding events during MPEG operation are ERR_EXTRA_PICTURE and ERR_EXTRA_SLICE.

When the Parser event is generated, the first byte of the extra information is presented through the register, rom_revision.

The state of the Video Demux register, continue, determines behavior after the event is cleared. If this register holds the value 0, then any remaining extra information will be consumed by the Video Demux and no events will be generated. If the continue is set to 1, an event will be generated as each byte of extra information arrives at the Video Demux. This continues until the extra information is exhausted or continue is set to 0.

NOTE:

1)The first byte of the extension/user data is always presented via the rom_revision

SECTION A.15 Spatial Decoding

In accordance with the present invention, the spatial decoding occurs between the output of the Token buffer and the output of the Spatial Decoder.

There are three main units responsible for spatial decoding: the inverse modeler, the inverse quantizer and the inverse discrete cosine transformer. At the input to this section (from the Token buffer) DATA Tokens contain a run and level representation of the quantized coefficients. At the output (of the inverse DCT) DATA Tokens contain 8x8 blocks of pixel information.

A.15.1 The Inverse Modeler

DATA Tokens in the Token buffer contain information about the values of quantized coefficients and the number of zeros between the coefficients that are represented. The Inverse Modeler expands the information about runs of zeros so that each DATA Token contains 64 values. At this point, the values in the DATA Tokens are quantized coefficients.

The inverse modelling process is the same regardless of the coding standard currently being used. No configuration is required.

For a better understanding of the modelling and inverse modelling function all requirements the reader can examine any of the picture coding standards.

A.15.2 Inverse Quantizer

In an encoder, the quantizer divides down the output of the DCT to reduce the resolution of the DCT coefficients. In a decoder, the function of the inverse quantizer is to multiply up these quantized DCT coefficients to restore them to an approximation of their original values.

A.15.2.1 Overview of the standard quantization schemes

There are significant differences in the quantization be used. However, use of the tables is quite different.

Two "types" of data are considered: intra and non-intra. A different table is used for each data type. Two "default" tables are defined by MPEG. One is for use with intra data and the other with non-intra data (see Table A.15.2 and Table A.15.3). These default tables must be written into the quantization table memory of the Spatial Decoder before MPEG decoding is possible.

MPEG also allows two "down loaded" quantization tables. One is for use with intra data and the other with non-intra data. The values for these tables are contained in the MPEG data stream and will be loaded into the quantization table memory automatically.

The value output from the tables is modified by a scale factor.

A.15.2.2 Inverse quantizer registers

Register name	Size/Dir.	Reset State	Description
iq_access	1 rw	0	This access bit stops the operation of the inverse quantiser so that its various registers can be accessed reliably. See A.6.4.1
iq_coding_standard	2 rw	0	This register configures the coding standard used by the inverse quantiser. The register can be loaded directly or by a CODING_STANDARD Token. See A.21.1
iq_keyhole_address	8 rw	x	Keyhole access to the which holds the 4 quantiser tables. See A.6.4.3 for more information about accessing registers through a keyhole.
iq_keyhole_data	8 rw	x	

Table A.15.1 Inverse quantizer registers

i^a	$W_{i,0}^b$	i	$W_{i,0}$	i	$W_{i,0}$	i	$W_{i,0}$
0	8	16	27	32	29	48	35
1	16	17	27	33	29	49	38
2	16	18	26	34	27	50	38
3	19	19	26	35	27	51	40
4	16	20	26	36	29	52	40
5	19	21	26	37	29	53	40
6	22	22	27	38	32	54	48
7	22	23	27	39	32	55	48
8	22	24	27	40	34	56	46
9	22	25	29	41	34	57	46
10	22	26	29	42	37	58	56
11	22	27	29	43	38	59	56
12	26	28	34	44	37	60	58
13	24	29	34	45	35	61	69
14	26	30	34	46	35	62	69
15	27	31	29	47	34	63	83

Table A.15.2 Default MPEG table for intra coded blocks

a. Offset from start of quantization table memory

b. Quantization table value.

The quantization table values are used in "zig-zag" scan order (see the coding standards). The tables should be viewed as a one dimensional array of 64 values (rather than a 8x8 array). The table entries at lower addresses correspond to the lower frequency DCT coefficients.

When quantization table values are carried by a QUANT_TABLE Token, the first value after the Token header is the table entry for the "DC" coefficient.

A.15.3 Inverse Discrete Cosine Transform

The inverse discrete transform processor of the present invention meets the requirements set out in CCITT recommendation H.261, the IEEE specification P1180 and complies with the requirements described in current draft revision of MPEG.

The inverse discrete cosine transform process is the same regardless of which coding standard is used. No, configuration by the user is required.

There are two events associated with the inverse discrete transform processor.

Register name	Size/Dir.	Reset State	Description
idct_too_few_event	1 rw	0	The inverse DCT requires that all DATA Tokens contain exactly 64 values. If less than 64 values are found then the too-few event will be generated. If the mask register is set to 1 then an interrupt can be generated and the inverse DCT will halt. This event should only occur following an error in the coded data.
idct_too_few_mask	1 rw	0	
idct_too_many_event	1 rw	0	The inverse DCT requires that all DATA Tokens contain exactly 64 values. If more than 64 values are found then the too-many event will be generated. If the mask register is set to 1 then an interrupt can be generated and the inverse DCT will halt. This event should only occur following an error in the coded data.
idct_too_many_mask	1 rw	0	

Table A.15.5 Inverse DCT event registers

For a better understanding of the DCT and inverse DCT function the reader can examine any of the picture coding standards.

A.16.1 Structure of JPEG pictures

This section provides an overview of some features of the JPEG syntax. Please refer to the coding standard for full details.

JPEG provides a variety of mechanisms for encoding individual pictures. JPEG makes no attempt to describe how a collection of pictures could be encoded together to provide a mechanism for encoding video.

The Spatial Decoder, in accordance with the present invention, supports JPEG's *baseline sequential* mode of operation. There are three main levels in the syntax: Image, Frame and Scan. A sequential image only contains a single frame. A frame can contain between 1 and 256 different image (color) components. These image components can be grouped, in a variety of ways, into scans. Each scan can contain between 1 and 4 image components (see Figure 81 "Overview of JPEG baseline sequential structure").

If a scan contains a single image component, it is *non-interleaved*, if it contains more than one image component, it is an *interleaved* scan. A frame can contain a mixture of interleaved and non-interleaved scans. The number of scans that a frame can contain is determined by the 256 limit on the number of image components that a frame can contain.

Within an interleaved scan, data is organized into minimum coding units (MCUs) which are analogous to the macroblock used in MPEG and H.261. These MCUs are raster ordered within a picture. In a non-interleaved scan, the MCU is a single 8x8 block. Again, these are raster organized.

The Spatial Decoder can readily decode JPEG data containing 1 to 4 different color components. Files describing greater numbers of components can also be

SECTION A.17 Temporal Decoder

- 30 MH$_z$ operation
- Provides temporal decoding for MPEG & H.261 video decoders
- H.261 CIF and QCIF formats
- MPEG video resolutions up to 704x480, 30 Hz, 4:2:0
- Flexible chroma sampling formats
- Can re-order the MPEG picture sequence
- Glue-less DRAM interface
- Single +5V supply
- 208 pin PQFP package
- Max. power dissipation 2.5W
- Uses standard page mode DRAM

The Temporal Decoder is a companion chip to the Spatial Decoder. It provides the temporal decoding required by H.261 and MPEG.

The Temporal Decoder implements all the prediction forming features required by MPEG and H.261. With a single 4 Mb DRAM (e.g., 512 k x 8) the Temporal Decoder can decode CIF and QCIF H.261 video. With 8 Mb of DRAM (e.g., two 256 k x 16) the 704 x 480, 30Hz, 4:2:0 MPEG video can be decoded.

The Temporal Decoder is not required for Intra coding schemes (such as JPEG). If included in a multi-standard decoder, the Temporal Decoder will pass decoded JPEG pictures through to its output.

Note: The above values are merely illustrative, by way of example and not necessarily by way of limitation, of one embodiment of the present invention. It will be appreciated that other values and ranges may also be used without departing from the invention.

Signal Name	I/O	Pin Num.	Description
tph0ish	I	122	If override = 1 then tph0ish and tph1ish are inputs for the on-chip two phase clock. For normal operation set override = 0. tph0ish and tph1ish are ignored (so connect to GND or V_{DD}).
tph1ish	I	123	
override	I	110	
chiptest	I	111	Set chiptest = 0 for normal operation.
tloop	I	114	Connect to GND or V_{DD} duing normal operation.
ramtest	I	109	If ramtest = 1 test of the on-chip RAMs is enabled. Set ramtest = 0 for normal operation.
pllselect	I	178	If pllselect = 0 the on-chip phase locked loops are disabled. Set pllselect = 1 for normal operation.
ti	I	180	Two clocks required by the DRAM interface during test operation. Connect to GND or V_{DD} during normal operation.
tq	I	179	
pdout	O	207	These two pins are connections for an external filter for the phase lock loop.
pdin	I	206	

Table A.17.2 Temporal Decoder Test signals

Signal Name	Pin	Signal Name	Pin	Signal Name	Pin	Signal Name	Pin
nc	208	nc	156	nc	104	nc	52
test pin	207	nc	155	nc	103	nc	51
test pin	206	irq	154	nc	102	nc	50
GND	205	nc	153	VDD	101	DRAM_data[15]	49
OE	204	data[7]	152	out_accept	100	nc	48
DRAM_addr[0]	203	data[6]	151	out_valid	99	DRAM_data[16]	47
VDD	202	nc	150	out_data[0]	98	nc	46
nc	201	data[5]	149	out_data[1]	97	GND	45
DRAM_addr[1]	200	nc	148	GND	96	DRAM_data[17]	44
DRAM_addr[2]	199	data[4]	147	out_data[2]	95	nc	43
GND	198	GND	146	out_data[3]	94	DRAM_data[18]	42
DRAM_addr[3]	197	data[3]	145	out_data[4]	93	VDD	41
nc	196	nc	144	out_data[5]	92	nc	40

Table A.17.3 Temp ral Dec der Pin Assignm nts

Signal Name	Pin	Signal Name	Pin	Signal Name	Pin	Signal Name	Pin
reset	160	nc	108	VDD	56	$\overline{\text{CAS}}$[2]	4
VDD	159	nc	107	nc	55	nc	3
nc	158	nc	106	nc	54	$\overline{\text{CAS}}$[3]	2
nc	157	nc	105	nc	53	nc	1

Table A.17.3 Temporal Decoder Pin Assignments (contd)

A.17.1.1 "nc" no connect pins

The pins labelled nc in Table A.17.3 are not currently used in the present invention and are reserved for future products. These pins should be left unconnected. They should not be connected to V_{DD}, GND, each other or any other signal.

A.17.1.2 V_{DD} and GND pins

As will be appreciated all the V_{DD} and GND pins provided must be connected to the appropriate power supply. The device will not operate correctly unless all the V_{DD} and GND pins are correctly used.

A.17.1.3 Test pin connections for normal operation

Nine pins on the Temporal Decoder are reserved for internal test use.

Pin number	Connection
	Connect to GND for normal operation
	Connect to V_{DD} for normal operation
	Leave Open Circuit for normal operation

Table A.17.4 Default test pin connections

Addr. (hex)	Bit num.	Register Name	Page references
0x08	7:1	not used	
	0	chip_access	

Table A.17.7 Chip access register

Addr. (hex)	Bit num.	Register Name	Page references
0x10	7:1	not used	
	0	MPEG_reordering	

Table A.17.8 Picture sequencing

Addr. (hex)	Bit num.	Register Name	Page references
0x28	7	not used	
	6:4	DRAM_addr_strength[3:0]	
	3:1	CAS_strength[3:0]	
	0	RAS_strength[3]	
0x29	7:6	RAS_strength[2:0]	
	5:3	OEWE_strength[3:0]	
	2:0	DRAM_data_strength[3:0]	
0x2A	7:0	refresh_interval	
0x2B	7:0	not used	
0x2C	7:6	not used	
	5	DRAM_enable	
	4	no_refresh	
	3:2	row_address_bits[1:0]	
	1:0	DRAM_data_width[1:0]	
0x2D	7:0	not used	
0x2E	7:0	Test registers	

Table A.17.9 DRAM interface configuration registers (contd)

Addr. (hex)	Bit num.	Register Name	Page references
0x40	7:0	not used	
0x41	7:2		
	1:0	picture_buffer_0[17:0]	
0x42	7:0		
0x43	7:0		
0x44	7:0	not used	
0x45	7:2		
	1:0	picture_buffer_1[17:0]	
0x46	7:0		
0x47	7:0		

Table A.17.10 Buffer configuration registers

Addr. (hex)	Bit num.	Register Name	Page references
0x64	7 ... 0	back_h	
0x65	7 ... 0		
0x66	7 ... 0	back_v	
0x67	7 ... 0		
0x68	7 ... 0	forw_h	
0x69	7 ... 0		
0x6A	7 ... 0	forw_v	
0x6B	7 ... 0		
0x6C	7 ... 0	width_in_mb	
0x6D	7 ... 0		

Table A.17.11 Test registers (contd)

Table A.17.11 Test registers (contd)

A.18.3.1 When to configure

The Temporal Decoder should only be configured when no data processing is taking place. This is the default state after reset is removed. The Temporal Decoder can be stopped to allow re-configuration by writing 1 to the chip_access register. After configuration is complete, 0 should be written to chip_access.

See Section A.5.3 for details of when to configure the DRAM interface.

A.18.3.2 DRAM interface

The DRAM interface timing must be configured before it is possible to decode predictively coded video (e.g., H.261 or MPEG). See Section A.5, "DRAM Interface".

A.18.~~3.~~3 Numbers in pictur buffer r gisters

The picture buffer pointers (18 bit) and the component offset (17 bit) registers specify a block (8x8 bytes) address, not a byte address.

A.18.3.4 Picture buffer allocation

To decode predictively coded video (either H.261 or MPEG) the Temporal Decoder must manage two picture buffers. See Section A.18.4 and A.18.4.4 for more information about how these buffers are used.

The user must ensure that there is sufficient memory above each of the picture buffer pointers (picture_buffer_0 and picture_buffer_1) to store a single picture of the required video format (without overlapping with the other picture buffer). Normally, one of the picture buffer pointers will be set to 0 (i.e., the bottom of memory) and the other will be set to point to the middle of the memory space.

A.18.3.4.1 Normal configuration for MPEG or H.261

H.261 and MPEG both use a 4:1:1 ratio between the different color components (i.e., there are 4 times as many luminance pels as there are pels in either of the chrominance components).

As documented in Section A.3.5.1, "Component Identification number", component 0 will be the luminance component and components 1 and 2 will be chrominance.

An example configuration of the component offset registers is to set component_offset_0 to 0 so that component 0 starts at the picture buffer pointer. Similarly, component_offset_1 could be set to 4/6 of the picture buffer size and component_offset_2 could be set to 5/6 of the picture buffer size.

A.18.3.5 Picture sequence re-ordering

MPEG uses three different picture types: Intra (I), operations involved, the reader is directed to the H.261 standard. The Temporal Decoder of the present invention is fully compliant with the requirements of H.261.

A.18.4.3 MPEG Operation (without re-ordering)

The operation of the Temporal Decoder changes for each of the three different MPEG picture types (I, P and B).

"I" pictures require no further decoding by the Temporal Decoder, but must be stored in a picture buffer (frame store) for later use in decoding P and B pictures.

Decoding P pictures requires forming predictions from a previously decoded P or I picture. The decoded P picture is stored in a picture buffer for use in decoding P and B pictures. MPEG allows motion vectors specified to half pixel accuracy. On-chip filters provide interpolation to support this half pixel accuracy.

B pictures can require predictions from both of the picture buffers. As with P pictures, half pixel motion vector resolution accuracy requires on chip interpolation of the picture information. B pictures are not stored in the off-chip buffers. They are merely transient.

All pictures appear at the output port of the Temporal Decoder as they are decoded. So, the picture sequence will be the same as that in the coded MPEG data (see the upper part of Figure 85).

For full details of prediction, and the arithmetic operations involved, the reader is directed to the proposed MPEG standard draft. These requirements are met by the Temporal Decoder of the present invention.

A.18.4.4 MPEG Operation (with re-ordering)

When configured for MPEG operation with picture re-ordering (MPEG_reordering = 1), the prediction forming operations are as described above in Section A.18.4.3. However, additional data transfers are performed to re-order the picture sequence.

lower memory bandwidth requirements. For example, using only integer resolution motion vectors or, alternatively, not using B pictures, significantly reduce the memory bandwidth requirements. Such subsets are not analyzed here.

A.18.5.1 Characteristics of DRAM interface

The number of cycles taken to transfer data across the DRAM interface depends on a number of factors:

- The timing configuration of the DRAM interface to suite the DRAM employed
- The data bus width (8, 16 or 32 bits)
- The type of data transfer:
  - 8x8 block read or write
  - for prediction to half pixel accuracy
  - for prediction to integer pixel accuracy

See section A.5, "DRAM Interface", for more information about the detail configuration of the DRAM interface.

Table A.18.3 shows how many DRAM interface "cycles" are required for each type of data transfer.

Data bus width (bits)	read or write 8x8 block	form prediction (half pixel accuracy)	form prediction (integer pixel accuracy)
8	1 page address + 64 transfers	4 page address + 81 transfers	4 page address + 64 transfers
16	1 page address + 32 transfers	4 page address + 45 transfers	4 page address + 40 transfers
32	1 page address + 16 transfers	4 page address + 27 transfers	4 page address + 24 transfers

Table A.18.3 Data transfer times for Temporal Decoder

If ~~the~~ required video format is 704 x 480, then each picture contains 7920 8 x 8 blocks (taking into consideration the 4:2:0 chroma sampling). It can be seen that this video format consumes approx. 91% of the available DRAM interface bandwidth (before any other factors such as DRAM refresh are taken into consideration). Accordingly, the Temporal Decoder can support this video format.

A.18.5.3 MPEG resolution with re-ordering

When MPEG picture re-ordering is employed the worst case scenario is encountered while P pictures are being decoded. During this time, there are 3 loads on the DRAM interface:

- form predictions
- write back the result
- read out the previous P or I picture

Using the example figures from Table A.18.3, we can find the time it takes for each of these tasks when a 32 bit wide interface is available. Forming the prediction takes 1907 ns/n while the read and the write each take 991 ns, a total of 3889 ns. This permits the Temporal Decoder to process 8485 8 x 8 blocks in a 33 ms period.

Hence, processing 704 x 480 video will use approximately 93% of the available memory bandwidth (ignoring refresh).

A.18.5.4 H.261

H.261 only supports two picture formats CIF (352 x 288) and QCIF (172 x 144) at picture rates up to 30 Hz. A CIF picture contains 2376 8 x 8 blocks. The only memory operations required are the writing of 8 x 8 blocks and the forming of predictions with integer accuracy motion vectors.

Using the example figures from Table A.18.4 for an 8 bit wide memory interface, it can be seen that writing each block will take 3657 ns while forming the prediction for one block will take 3963 ns/n, a total of 7620 ns per

SECTION A.19 Connecting to the output of the Temporal Decoder

The output of the Temporal Decoder is a standard Token Port with 8 bit wide data words. See Section A.4 for more information about the electrical behavior of the interface.

The Tokens present at the output of the Temporal Decoder will depend on the coding standard employed and, in the case of MPEG, whether the pictures are being re-ordered. This section identifies which of the Tokens are available at the output of the Temporal decoder and which are the most useful when designing circuits to display that output. Other Tokens will be present, but are not needed to display the output and, therefore they are not discussed here.

This section concentrates on showing:

- How the start and end of sequences can be identified.
- How the start and end of pictures can be identified.
- How to identify when to display the picture.
- How to identify where in the display the picture data should be placed.

A.19.1 JPEG output

The Token *sequence* output by the Temporal Decoder when decoding JPEG data is identical to that seen at the output of Spatial Decoder. Recall, JPEG does not require processing by the Temporal Decoder. However, the Temporal Decoder tests intra data Tokens for negative values (resulting from the finite arithmetic precision of the IDCT in the Spatial Decoder) and replaces them with zero.

See Section A.16 for further discussion of the output sequence observed during JPEG operation.

TEMPORAL_REFERENCE Token carries a 10 bit number (of which only the 5 LSBs are used in H.261) that indicates when the picture should be displayed. This should be studied by any display system as H.261 encoders can omit pictures from the sequence (to achieve lower data rates). Omission of pictures can be detected by the temporal reference incrementing by more than one between successive pictures.

Next, the PICTURE_TYPE Token carries information about the picture format. A display system may study this information to detect if CIF or QCIF pictures are being decoded. However, information about the picture format is also available by studying registers within the Huffman decoder.

<Xref to Huffman decoder section>

A.19.2.2.2 Group of Blocks Layer

Each H.261 picture is composed of a number of "groups of blocks". Each of these is preceded by a SLICE_START Token (derived from the H.261 group number and group start code). This Token carries an 8 bit value that indicates where in the display the group of blocks should be placed. This provides an opportunity for the decoder to resynchronize after data errors. Moreover, it provides the encoder with a mechanism to skip blocks if there are areas of a picture that do not require additional information in order to describe them. By the time SLICE_START reaches the output of the Temporal Decoder, this information is effectively redundant as the Spatial Decoder and Temporal Decoder have already used the information to ensure that each picture contains the correct number of blocks and that they are in the correct positions. Hence, it should be possible to compute where to position a block of data output by the Temporal Decoder just by counting the number of blocks that have been output since the start of the picture.

The number carried by SLICE_START is one less than the

A.19.2.2.3 Macroblock layer

The sequence of macroblocks within each group of blocks is defined by H.261. There is no special Token information describing the position of each macroblock. The user should count through the macroblock sequence to determine where to display each piece of information.

Figure 96 shows the sequence in which macroblocks are placed in each group of blocks.

Each macroblock contains 6 DATA Tokens. The sequence of DATA Tokens in each group of 6 is defined by the H.261 macroblock structure. Each DATA Token should contain exactly 64 data bytes for an 8x8 area of pixels of a single color component. The color component is carried in a 2 bit number in the DATA Token (see section A.3.5.1). However, the sequence of the color components in H.261 is defined.

Each group of DATA Tokens is preceded by a number of Tokens communicating information about motion vectors, quantizer scale factors and so forth. These Tokens are not required to allow the pictures to be displayed and, thus, can be ignored.

Each DATA Token contains 64 data bytes for an 8x8 of a single color component. These are in a raster order.

A.19.3 MPEG output

MPEG has more layers in its syntax. These embody concepts such as a video sequence and the group of pictures.

A.19.3.1 MPEG Sequence layer

A sequence can have multiple entry points (sequence starts) but should have only a single exit point (sequence end). When an MPEG sequence header code is decoded, the Spatial Decoder generates a CODING_STANDARD Token followed by a SEQUENCE_START Token.

After the SEQUENCE_START, there will be a number of

A.19.3.3. Picture layer

The start of a new picture is indicated by the PICTURE_START Token. After this Token, there will be TEMPORAL_REFERENCE and PICTURE_TYPE Tokens. The temporary reference information may be useful if the Temporal Decoder is not configured to provide picture re-ordering. The picture type information may be useful if a display system wants to specially process B pictures at the start of an open GOP (see Section A.19.3.8).

Each picture is composed of a number of slices.

A.19.3.4 Slice layer

Section A.19.2.2.2 discusses the group of blocks used in H.261. The slice in MPEG serves a similar function. However, the slice structure is not fixed by the standard. The 8 bit value carried by the SLICE_START Token is one less than the "slice vertical position" communicated by MPEG. See the draft MPEG standard for a description of the slice layer.

By the time SLICE_START reaches the output of the Temporal Decoder, this information is effectively redundant since the Spatial Decoder and Temporal Decoder have already used the information to ensure that each picture contains the correct number of blocks in the correct positions. Hence, it should be possible to compute where to position a block of data output by the Temporal Decoder just by counting the number of blocks that have been output since the start of the picture.

See section A.19.3.7 for discussion of the effects of using MPEG picture re-ordering.

A.19.3.5 Macroblock layer

Each macroblock contains 6 blocks. These appear at the output of the Temporal Decoder in raster order (as specified by the draft MPEG specification).

display it picks up the lower level non-DATA tokens of the picture that has just been decoded. Hence, these sub-picture layer Tokens should be ignored.

A.19.3.8 Random access and edited sequences

The Spatial Decoder provides facilities to help correct video decoding of edited MPEG video data and after a random access into MPEG video data.

A.19.3.8.1 Open GOPs

A group of pictures (GOP) can start with B pictures that are predicted from a P picture in a previous GOP. This is called an "open GOP". Figure 107 illustrates this. Pictures 17 and 18 are B pictures at the start of the second GOP. If the GOP is "open", then the encoder may have encoded these two pictures using predictions from the P picture 16 and also the I picture 19. Alternatively, the encoder could have restricted itself to using predictions from only the I picture 19. In this case, the second GOP is a "closed GOP".

If a decoder starts decoding the video at the first GOP, it will have no problems when it encounters the second GOP even if that GOP is open since it will have already decoded the P picture 16. However, if the decoder makes a random access and starts decoding at the second GOP it cannot decode B17 and B18 if they depend on P16 (i.e., if the GOP is open).

If the Spatial Decoder of the present invention encounters an open GOP as the first GOP following a reset or it receives a FLUSH Token, it will assume that a random access to an open GOP has occurred. In this case, the Huffman decoder will consume the data for the B pictures in the normal way. However, it will output B pictures predicted with (0,0) motion vectors off the I picture. The result will be that pictures B17 and B18 (in the example above) will be identical to I19.

or "substitutes" that have been introduced by the Spatial Decoder. Some applications may wish to take special measures when these "substitute" pictures are present.

Register name	Size/ Dir.	Reset State	Description
page_start_length	5 bit rw	0	Specifies the length of the access start in ticks. The minimum value that can be used is 4 (meaning 4 ticks). 0 selects the maximum length of 32 ticks.
read_cycle_length	4 bit rw	0	Specifies the length of the fast page read cycle in ticks. The minimum value that can be used is 4 (meaning 4 ticks). 0 selects the maximum length of 16 ticks.
write_cycle_length	4 bit rw	0	Specifies the length of the fast page late write cycle in ticks. The minimum value that can be used is 4 (meaning 4 ticks). 0 selects the maximum length of 16 ticks.
refresh_cycle_length	4 bit rw	0	Specifies the length of the refresh cycle in ticks. The minimum value that can be used is 4 (meaning 4 ticks). 0 selects the maximum length of 16 ticks.
RAS_falling	4 bit rw	0	Specifies the number of ticks after the start of the access start that $\overline{\text{RAS}}$ falls. The minimum value that can be used is 4 (meaning 4 ticks). 0 selects the maximum length of 16 ticks.
CAS_falling	4 bit rw	8	Specifies the number of ticks after the start of a read cycle, write cycle or access start that $\overline{\text{CAS}}$ falls. The minimum value that can be used is 1 (meaning 1 tick). 0 selects the maximum length of 16 ticks.
DRAM_data_width	2 bit rw	0	Specifies the number of bits used on the DRAM interface data bus DRAM_data[31:0]. See A.20.4
row_address_bits	2 bit rw	0	Specifies the number of bits used for the row address portion of the DRAM interface address bus. See A.20.5
DRAM_enable	1 bit rw	1	Writing the value 0 in to this register forces the DRAM interface into a high impedance state. 0 will be read from this register if either the DRAM_enable signal is low or 0 has been written to the register.

Table A.20.2 DRAM Interface configuration registers (contd)

A.20.1 Interface timing (ticks)

In the present invention, the DRAM interface timing is derived from a clock which is running at four times the input clock rate of the device (decoder_clock). This clock is generated by an on-chip PLL.

For brevity, periods of this high speed clock are referred to as *ticks*.

A.20.2 Interface operation

The interface uses of the DRAM fast page mode. Three different types of access are supported:

- Read
- Write
- Refresh

Each read or write access transfers a burst of between 1 and 64 bytes at a single DRAM page address. Read and write transfers are not mixed within a single access. Each successive access is treated as a random access to a new DRAM page.

A.20.3 Access structure

Each access is composed of two parts:

- Access start
- Data transfer

Each access starts with an access start and is followed by one or more *data transfer* cycles. There is a read, write and refresh variant of both the access start and the *data transfer* cycle.

At the end of the last data transfer in an access the interface enters it's *default state* and remains in this state until a new access is ready to start. If a new access is ready to start when the last access finishes, then the new access will start immediately.

A.20.3.1 Access start

The *access start* provides the page address for the read or write transfers and establishes some initial signal

A.20.3.2. Data transfer

There are three different types of data transfer cycle:

- Fast page read cycle
- Fast page late write cycle
- Refresh cycle

A start of refresh is only followed by a single refresh cycle. A start of read (or write) can be followed by one or more fast page read (or write) cycles.

At the start of the read cycle $\overline{\textbf{CAS}}$ is driven high and the new column address is driven.

A late write cycle is used. **WE** is driven low one tick after $\overline{\textbf{CAS}}$. The output data is driven one tick after the address.

As a $\overline{\textbf{CAS}}$ before $\overline{\textbf{RAS}}$ refresh cycle is initiated by the start of refresh cycle, there is no interface signal activity during a refresh cycle. The purpose of the refresh cycle is to meet the minimum $\overline{\textbf{RAS}}$ low period required by the DRAM.

A.20.3.3 Interface default state

The interface signals enter a default state at the end of an access:

- $\overline{\textbf{RAS}}$, $\overline{\textbf{CAS}}$ and **WE** high
- data and **OE** remain in their previous state
- addr remains stable

A.20.4 Data bus width

The two bit register DRAM_data_width allows the width of the DRAM interfaces data path to be configured. This allows the DRAM cost to be minimized when working with small picture formats.

A.20.~~5~~.2 Row address bits

The number of bits taken from the middle section of the 24 bit internal address to provide the row address is configured by the register row_address_bits.

row_address_bits	Width of row address
0	9 bits
1	10 bits
2	11 bits

Table A.20.5 Configuring row_address_bits

The width of row address used will depend on the type of DRAM used and whether the MSBs of the row address are decoded off-chip to access multiple banks of DRAM.

NOTE: The row address is extracted from the middle of the internal address. If some bits of the row address are decoded to select banks of DRAM, then all possible values of these "bank select bits" must select a bank of DRAM. Otherwise, holes will be left in the address space.

Decoder) is not in use. It is not intended to allow other devices to share the memory during normal operation.

A.20.7 R fr sh

Unless disabled by writing to the register, no_refresh, the DRAM interface will automatically refresh the DRAM using a $\overline{\text{CAS}}$ before $\overline{\text{RAS}}$ refresh cycle at an interval determined by the register refresh_interval.

The value in refresh_interval specifies the interval between refresh cycles in periods of 16 decoder_clock cycles. Values in the range 1 to 255 can be configured. The value 0 is automatically loaded after reset and forces the DRAM interface to continuously execute refresh cycles (once enabled) until a valid refresh interval is configured. It is recommended that refresh_interval should be configured *only* once after each reset.

A.20.8 Signal strengths

The drive strength of the outputs of the DRAM interface can be configured by the user using the 3 bit registers, CAS_strength, RAS_strength, addr_strength, DRAM_data_strength, OEWE_strength. The MSB of this 3 bit value selects either a fast or slow edge rate. The two less significant bits configure the output for different load capacitances.

The default strength after reset is 6, configuring the outputs to take approximately 10 ns to drive signal between GND and V_{DD} if loaded with 12_pF.

after power is first applied, followed by a number of refresh cycles before normal operation is possible.

Immediately after reset, the DRAM interface is inactive until both the DRAM_enable signal and the DRAM_enable register are set. When these have been set, the DRAM interface will execute refresh cycles (approximately every 400 ns, depending upon the clock frequency used) until the DRAM interface is configured.

The user is responsible for ensuring that the DRAM's "pause" after power_up and for allowing sufficient time after enabling the DRAM interface to ensure that the required number of refresh cycles have occurred before data transfers are attempted.

While reset is asserted, the DRAM interface is unable to refresh the DRAM. However, the reset time required by the decoder chips is sufficiently short so that is should be possible to reset them and to then re-enable the DRAM interface before the DRAM contents decay. This may be required during debugging.

Symbol	Parameter	Min.	Max.	Units
V_{DD}	Supply voltage relative to GND	-0.5	6.5	V
V_{IN}	Input voltage on any pin	GND - 0.5	V_{DD} + 0.5	V
T_A	Operating temperature	-40	+85	°C
T_S	Storage temperature	-55	+150	°C

Table A.20.8 Maximum Ratings[a]

A.20.10.1 AC charact ristics

Num.	Parameter	Min.	Max.	Unit	Note [a]
45	Cycle time e.g. tPC	-2	+2	ns	
46	Cycle time e.g. tRC	-2	+2	ns	
47	High pulse e.g. tRP, tCP, tCPN	-5	+2	ns	
48	Low pulse e.g. tRAS, tCAS, tCAC, tWP, tRASP, tRASC	-11	+2	ns	
49	Cycle time e.g. tACP/tCPA	-8	+2	ns	

Table A.20.11 Differences from nominal values for a strobe

Table A.20.11 Differences from nominal values for a strob

a. The driver strength of the signal must be configured appropriately for its load

Num.	Parameter	Min.	Max.	Unit	Note [a]
50	Strobe to strobe delay e.g. tRCD, tCSR	-3	+3	ns	
51	Low hold time e.g. tRSH, tCSH, tRWL, tCWL, tRAC, tOAC/OE, tCHR	-13	+3	ns	
52	Strobe to strobe precharge e.g. tCRP, tRCS, tRCH, tRRH, tRPC	-9	+3	ns	
	$\overline{CAS}$ precharge pulse between any two $\overline{CAS}$ signals on wide DRAMs e.g. tCP, or between $\overline{RAS}$ rising and $\overline{CAS}$ falling e.g. tRPC	-5	+2	ns	

Table A.20.12 Differences from nominal values between two strobes

Table A.20.12 Differences from nominal values between two strobes

SECTION B.1 Start Code Detector

B.1.1 Ov rview

As previously shown in Figure 11, the Start Code Detector (SCD) is the first block on the Spatial Decoder. Its primary purpose is to detect MPEG, JPEG and H.261 start codes in the input data stream and to replace them with relevant Tokens. It also allows user access to the input data stream via the microprocessor interface, and performs preliminary formatting and "tidying up" of the token data stream. Recall, the SCD can receive either raw byte data or data already assembled in Token format.

Typically, start codes are 24, 16 and 8 bits wide for MPEG, H.261, and JPEG, respectively. The Start Code Detector takes the incoming data in bytes, either from the Microprocessor Interface (upi) or a token/byte port and shifts it through three shift registers. The first register is an 8 bit parallel in serial out, the second register is of programmable length (16 or 24 bits) and is where the start codes are detected, and the third register is 15 bits wide and is used to reformat the data into 15 bit tokens. There are also two "tag" Shift Registers (SR) running parallel with the second and third SRs. These contain tags to indicate whether or not the associated bit in the data SR is good. Incoming bytes that are not part of a DATA Token and are unrecognized by the SCD, are allowed to bypass the shift registers and are output when all three shift registers are flushed (empty) and the contents output successfully. Recognized non-data tokens are used to configure the SCD, spring traps, or set flags. They also bypass the shift registers and are output unchanged.

B.1.2 Major Blocks

The hardware for the Start Code Detector consists of 10 state machines.

B.1.2.1 Input Circuit (scdipc.sch.iplm.M)

The input circuit has three modes of operation: token, byte and microprocessor interface. These modes allow data

Table B.1.1. R cogniz d input tokens

Input Token	Command issued	Comments
NULL	WAIT	NULLs are removed
DATA	NORMAL	Load next byte into first SR
CODING_STD	BYPASS	Flush shift registers, perform padding, output and switch to bypass mode. Load CODING_STANDARD register.
FLUSH	BYPASS	Flush SRs with padding, output and switch to bypass mode.
ELSE (unrecognised token)	BYPASS	Flush SRs with padding, output and switch to bypass mode.

Note: A change in coding standard is passed to all blocks via the two-wire interface after the SRs are flushed. This ensures that the change from one data stream to another happens at the correct point throughout the SCD. This principle is applied throughout the presentation so that a change in the coding standard can flow through the whole chip prior to the new stream.

B.1.2.3 JPEG (scdjpeg.sch scdjpegm.M)

Start codes (Markers) in JPEG are sufficiently different that JPEG has a state machine all to itself. In the present invention, this block handles all the JPEG marker detection, length counting/checking, and removal of data. Detected JPEG markers are flagged as start codes (with v_not_t - see later text) and the command from scdipnew is overridden and forced to bypass. The operation is best described in code.

```
switch (state)
{
  case (LOOKING):
    if (input == 0xff)
    {
      state = GETVALUE; /*Found a marker*/
      remove; /*Marker gets removed*/
    }
    else
```

```
     state = CHECKLC;
     break;
     case (CHECKLC):
       if (lcnt == 0)
         state = LOOKING;/*No more to do*/
       else if (lcnt < 0)
         state = LOOKING;/*generate Illegal_Length_Error*/
       else
       state = COUNT;
    break;
    case (COUNT):
      decrement length count until 1
      if (lc <= 1)
        state = LOOKING;
}
```

B.1.2.6 Output Shifter (scoshift.sch, scoshm.M)

The basic operation of the output shifter is to take serial data (and tags) from scdetect, pack it into 15 bit words and output them. Other functions are:

B.1.2.6.1 Data padding

The output consists of 15 bit words, but the input may consist of an arbitrary number of bits. In order to flush, therefore,we need to add bits to make the last word up to 15 bits. These extra bits are called padding and must be recognized and removed by the Huffman block. Padding is defined to be:

After the last data bit, a "zero" is inserted followed by sufficient "ones" to make up a 15 bit word.

The data word containing the padding is output with a low extension bit to indicate that it is the end of a data token.

B.1.2.6.2 Generation of "flushed"

In accordance with the present invention, the generation of "flushed" operation involves detecting when all SRs are flushed and signalling this to the input shifter. When the "rubbish" inserted by the input shifter reaches the end of the output shifter, and the output shifter has completed its padding, a "flushed" signal is generated. This "flushed" signal must pass through the token reconstructor before it is safe for the input shifter to enter bypass mode.

B.1.2.6.3 Flagging valid start codes

If scdetect indicates that it has found a start code, padding is performed and the current data is output. The start code value (the next byte) is shifted through the detector to eliminate overlapping start codes. If the "value" arrives at the output shifter without another start code being detected, it was not overlapped and the value is passed out with a flag v_not_t (ValueNotToken) to indicate that it is a start code value. If, however, another start code is detected (by scdetect) whilst the output shifter is waiting for the value, an overlapping_start_error is

Tabl B.1.2 Start Cod numbers (indices)

Start/Marker Code	Index (start_number)	Resulting Token
not_a_start_code	0	--
sequence_start_code	1	SEQUENCE_START
group_start_code	2	GROUP_START
picture_start_code	3	PICTURE_START
slice_start_code	4	SLICE_START
user_data_start_code	5	USER_DATA
extension_start_code	6	EXTENSION_DATA
sequence_end_code	7	SEQUENCE_END
JPEG Markers		
DHT	8	DHT
DQT	9	DQT
DNL	10	DNL
DRI	11	DRI
JPEG markers that can be mapped onto tokens for MPEG/H.261		
SOS	picture_start_code	PICTURE_START
SOI	sequence_start_code	SEQUENCE_START

Table B.1.2 Start Code numbers (indices)

Start/Marker Code	Index (start_number)	Resulting Token
EOI	sequence_end_code	SEQUENCE_END
SOF0	group_start_code	GROUP_START
JPEG markers that generate extn or user data		
JPG	extension_start_code	EXTENSION_DATA
JPGn	extension_start_code	EXTENSION_DATA
APPn	user_data_start_code	USER_DATA
COM	user_data_start_code	USER_DATA
NOTE: All unrecognised JPEG markers generate an extn_start_code index		

B.1.2.9 Start number to token conversion (sconvert.sch, sconverm.M)

The second stage of the conversion is where the above start numbers (or indices) are converted into tokens. This block also handles token extensions where appropriate, discarding of extension and user data, and search modes.

```
switch (input_data)
 case (FLUSH)
  1. if (in_picture)
      output = PICTURE_END
  2. output = FLUSH
  3. if (in_picture & stop_after_picture)
      sap_error = HIGH
      in_picture = FALSE;
  4. in_picture = FALSE;
 break
 case (SEQUENCE_START)
  1. if (in_picture)
      output = PICTURE_END
  2. if (in_picture & stop_after_picture)
      2a. output = FLUSH
      2b. sap_error = HIGH
          in_picture = FALSE
  3. output = CODING_STANDARD
  4. output = standard
  5. output = SEQUENCE_START
  6. in_picture = FALSE;
 break
 case (SEQUENCE_END) case (GROUP_START):
  1. if (in_picture)
      output = PICTURE_END
  2. if (in_picture & stop_after_picture)
      2a. output = FLUSH
      2b. sap_error = HIGH
          in_picture = FALSE
  3. output = SEQUENCE_END  r GROUP_START
  4. in_picture = FALSE;
break
 case (PICTURE_END)
```

SECTION B.2 Huffman Decoder and Parser

B.2.1 Introduction

This section describes the Huffman Decoder and Parser circuitry in accordance with the present invention.

Figure 118 shows a high level block diagram of the Huffman Decoder and Parser. Many signals and buses are omitted from this diagram in the interests of clarity, in particular, there are several places where data is fed backwards (within the large loop that is shown).

In essence, the Huffman Decoder and Parser of the present invention consist of a number of dedicated processing blocks (shown along the bottom of the diagram) which are controlled by a programmable state machine.

Data is received from the Coded Data Buffer by the "Inshift" block. At this point, there are essentially two types of information which will be encountered: Coded data which is carried by DATA Tokens and start codes which have already been replaced by their respective Tokens by the Start Code Detector. It is possible that other Tokens will be encountered but all Tokens (other than the DATA Tokens) are treated in the same way. Tokens (start codes) are treated as a special case as the vast majority of the data will still be encoded (in H.261, JPEG or MPEG).

In the present invention, all data which is carried by the DATA Tokens is transferred to the Huffman Decoder in a serial form (bit-by-bit). This data, of course, includes many fields which are not Huffman coded, but are fixed length coded. Nevertheless, this data is still passed to the Huffman Decoder serially. In the case of Huffman encoded data, the Huffman Decoder only performs the first stage of decoding in which the actual Huffman code is replaced by an index number. If there are N district Huffman codes in the particular code table which is being decoded, then this "Huffman Index" lies in the range 0 to N-1. Furthermore, the Huffman Decoder has a "no op", i.e., "no operation" mode, which allows it to pass along data or token information to a subsequent stage without any by two-wire interfaces.

In the present invention, there is a two-wire interface between each of the blocks in the Video Parser. Furthermore, the Huffman Decoder works with both serial, data, the inshifter inputs data one bit at a time, and with control tokens. Accordingly, there are two modes of operation. If data is coming into the Huffman Decoder via a DATA Token, then it passes through the shifter one bit at a time. Again, there is a two-wire interface between the inshifter and the Huffman Decoder. Other tokens, however, are not shifted in one bit at a time (serial) but rather in the header of the token. If a DATA token is input, then the header containing the address information is deleted and the data following the address is shifted in one bit at a time. If it is not a DATA Token, then the entire token, header and all, is presented to the Huffman Decoder all at once.

In the present invention, it is important to understand that the two-wire interface for the Video Parser is unusual in that it has two valid lines. One line is valid serially and one line is valid tokenly. Furthermore, both lines may not be asserted at the same time. One or the other may be asserted or if no valid data exists, then neither may be asserted although there are two valid lines, it should be recognized that there is only a single accept wire in the other direction. However, this is not a problem. The Huffman Decoder knows whether it wants serial data or token information depending on what needs to be done next based upon the current syntax. Hence, the valid and accept signals are set accordingly and an Accept is sent from the Huffman Decoder to the inshifter. If the proper data or token is there, then the inshifter sends a valid signal.

For example, a typical instruction might decode a Huffman code, transform it in the Index to Data Unit, modify that result in the ALU and then this result is formed into a Token word. A single microcode instruction word is produced which contains all of the information to do this.

Formatter takes this information and puts it into a token for use by the rest of he system. Note that the number of bits from each source in the above examples are merely for illustration purposes and one of ordinary skill in the art will appreciate that the number of bits from either source can vary.

The ALU includes a bank of counters that are used to count through the structure of the picture. The dimensions of the picture are programmed into registers associated with the counters that appear to the "microprogrammer" as part of the register bank. Several of the condition codes are outputs from this counter bank which allows conditional jumps based on "start of picture", "start of macroblock" and the like.

Note that the Parser State Machine is also referred to as the "Demultiplex State Machine". Both terms are used in this document.

Input Shifter

In the present invention, the Input Shifter is a very simple piece of circuitry consisting of a two pipeline stage datapath ("hfidp") and controlling Zcells ("hfi").

In the first pipeline stage, Token decoding takes place. At this stage, only the DATA token is recognized. Data contained in a DATA token is shifted one bit at a time into the Huffman Decoder. The second pipeline stage is the shift register. In the very last word of a DATA token, special coding takes place such that it is possible to transmit an arbitrary number of bits through the coded data buffer. The following are all possible patterns in the last data word.

B.2.2 Huffman D cod r

The Huffman Decoder has a number of mod s of operation. The most obvious is that it can decode Huffman Codes, turning them into a Huffman Index Number. In addition, it can decode fixed length codes of a length (in bits) determined by the instruction word. The Huffman Decoder can also accept Tokens from the Inshift block.

The Huffman Decode includes a very small state machine. This is used when decoding block-level information. This is because it takes too long for the Parser State Machine to make decisions (since it must wait for data to flow through the Index to Data Unit and the ALU before it can make a decision about that data and issue a new command). When this State Machine is used, the Huffman Decoder its lf issues commands to the Index to Data Unit and ALU. The Huffman Decoder State Machine cannot control all of the microcode instruction bits and, therefore, it cannot issue the full range of commands to the other blocks.

B.2.2.1 Theory of Operation

When decoding Huffman codes, the Huffman Decoder of the present invention uses an arithmetic procedure to decode the incoming code into a Huffman Index Number. This number lies between 0 and N-1 (for a code table that has N entries). Bits are accepted one by one from the Input shifter.

In order to control the operation of the machine, a number of tables are required. These specify for each possible number of bits in a code (1 to 16 bits) how many codes there are of that length. As expected, this information is typically not sufficient to specify a general Huffman code. However, in MPEG, H.261 and JPEG, the Huffman codes are chosen such that this information alone can specify the Huffman Code table. There is unfortunately just one exception to this; the Tcoefficient table from H.261 which is also used in MPEG. This r quires an additional table that is described elsewhere (the exception was deliberately introduced in H.261 to avoid

EQ 1. $\text{total}_{n+1} = \text{total}_n + \text{cpb}_n$

EQ 2. $'s_{n+1} = 2('s_n + \text{cpb}_n)$

EQ 3. $\text{code}_{n+1} = 2\text{code}_n + \text{bit}_n$

EQ 4. $\text{index}_{n+1} = 2\text{code}_n + \text{bit}_n + \text{total}_n - 's_n$

Unfortunately in the hardware it proved easier to use a modified set of equations in whic
a variable "shifted" is used in place of the variable "s". In this case;

In the hardware, however, it proved easier to use a modified set of equations in which a variable "shifted" is used in place of the variable "s". In this case;

EQ 5. $\text{shifted}_{n+1} = 2\text{shifted}_n + \text{cpb}_n$

It turns out that:

EQ 6. $;_n = 2\text{shifted}_n$

and so substituting this back into Equation 4 we see that:

EQ 7. $\text{index}_{n+1} = 2(\text{code}_n - \text{shifted}_n) + \text{total}_n + \text{bit}_n$

In addition to calculating successive values of "index", it is necessary to know when the calculation is completed. From the "C" code fragment we see that we are done when:

EQ 8. $\text{index}_{n+1} < \text{total}_{n+1}$

those inherited from JPEG (the decoding of DC intra coefficients) the JPEG convention is used.

B.2.2.1.2 Transform C efficients Table

When using the transform coefficients table in H.261 and MPEG, there are number of anomalies. First, the table in MPEG is a super-set of the table in H.261. In the hardware implementation of the present invention, there is no distinction drawn between the two standards and this means that an H.261 stream that contains codes from the extended part of the table (i.e., MPEG codes) will be decoded in the "correct" manner. Of course, other aspects of the compression standard may well be broken. For example, these extended codes will cause start code emulation in H.261.

Second, the transform coefficient table has an anomaly that means that it is not describable in the normal manner with the codes_per_bit tables. This anomaly occurs with the codes of length six bits. These code words are systematically substituted by alternate code words. In an encoder, the correct result is obtained by first encoding in the normal manner. Then, for all codes that are six bits or longer, the first six bits are substituted by another six bits by a simple table look-up operation. In a decoder, in accordance with the present invention, the decoding process is interrupted just before the sixth bit is decoded, the code words are substituted using a table look-up, and the decoding continues.

In this case, there are only ten possible six-bit codes so the necessary look-up table is very small. The operation is further helped by the fact that the upper two bits of the code are unaltered by the operation. As a result, it is not necessary to use a true look-up table. Instead a small collection of gates are hard-wired to give the appropriate transformation. The module that does this is called "hftcfrng". This type of code substitution is defined herein as a "ring" since each code from the set of possible codes is replaced by another code from that set

Twelve bits of "code" are preserved. This is not necessary for decoding Huffman codes where an eight bit register would have been sufficient. These upper bits are required for fixed length codes where up to twelve bits may be read.

B.2.2.1.4 Operation for Fixed Length Codes

For fixed length codes, the "codes per bit" value is forced to zero. This means that "total" and "shifted" remain at zero throughout the operation and "index" is, therefore, the same as code. In fact, the adders and the like only allow an eight bit value to be produced for "index". Because of this, the upper bits of the output word are taken directly from the "code" register when decoding fixed length codes. When decoding Huffman codes these upper bits are forced to zero.

The fact that sufficient bits have been read from the input is calculated in the obvious manner. A comparator compares the desired number of bits with the "bit" counter.

B.2.2.2 Decoding Coefficient Data

The Parser State Machine, in accordance with the present invention, is generally only used for fairly high-level decoding. The very lowest level decoding within an eight-by-eight block of data is not directly handled by this state machine. The Parser State Machine gives a command to the Huffman Decoder of the form "decode a block". The Huffman Decoder, Index to Data Unit and ALU work together under the control of a dedicated state machine (essentially in the Huffman Decoder). This arrangement allows very high performance decoding of entropy coded coefficient data. There are also other feedback paths operational in this mode of operation. For instance, in JPEG decoding where the VLCs are decoded to provide SIZE and RUN information, the SIZE information is fed back directly from the output of the Index to Data Unit to the Huffman Decoder to instruct the Huffman Decoder how many FLC bits to read. In addition, there are several accelerators implemented. For instance, using the same example all VLC values which yield explicitly read in this implementation because it is necessary to send different commands to the ALU for negative values versus positive values. This allows the ALU to convert the twos complement value in the bit stream into sign magnitude. In either case, the remaining seven bits of FLC are then read. If this has the value zero, then a further eight bits must be read.

In the present invention, the Huffman Decoder's internal state machine is responsible for generating commands to control itself and to also control the Index to Data Unit, the ALU and the Token Formatter. As shown in Figure 124, the Huffman Decoder's instruction comes from one of three sources, the Parser State Machine, the Huffman State Machine or an instruction stored in a register that has previously been received from the Parser State Machine. Essentially, the original instruction from the Parser State Machine (that causes the Huffman State Machine to take over control and read coefficients) is retained in a register, i.e., each time a new VLC is required, it is used. All the other instructions for the decoding are supplied by the Huffman State Machine.

B.2.2.2.2 MPEG DC Coefficient Data

This is handled in the same way as JPEG DC Coefficient Data. The same (loadable) tables are used and it is the responsibility of the controlling microprocessor to ensure that their contents are correct. The only real difference from the MPEG standard is that the predictors are reset to zero (like in JPEG) the correction for this being made in the Inverse Quantizer.

B.2.2.2.3 JPEG Coefficient Data

Figure 120 is a block diagram illustrating the hardware, in accordance with the present invention, for decoding JPEG AC Coefficients. Since the process for DC Coefficients is essentially a simplication of the JPEG process, the diagram serves for both AC and DC Coefficients. The only real addition to the previous diagram for the MPEG AC coefficients is that the "SSSS" first sign-extending the value with the *wrong* sign. If the sign bit is now set, then the remaining bits are inverted (ones complement).

In the case of DC Coefficients, the decision making in the Huffman Decoding Stage is somewhat easier because there is no equivalent of the ZRL field. The only symbol which causes zero FLC bits to be read is the one indicating zero DC difference. This is again trapped at the Huffman Index stage, a register being provided to hold this index for each of the (downloadable) JPEG DC tables.

The ALU of the present invention has the job of forming the final decoded DC coefficient by retaining a copy of the last DC Coefficient value (known as the prediction). Four predictors are required, one for each of the four active color components. When the DC difference has been decoded, the ALU adds on the appropriate predictor to form the decoded value. This is stored again as the predictor for the next DC difference of that color component. Since DC coefficients are signed (because of the DC offset) conversion from twos complement to sign magnitude is required. The value is then output with a RUN of zero. In fact, the instructions to perform some of the last stages of this are not supplied by the Huffman State Machine. They are simply executed by the Parser State Machine.

In a similar manner to the AC Coefficients, the ALU must first form the DC difference from the SIZE bits of FLC. However, in this case, a twos complement value is required to be added to the predictor. This can be formed by first sign extending with the wrong sign, as before. If the result is negative, then one must be added to form the correct value. This can, of course, be added at the same time as the predictor by jamming the carry into the adder.

B.2.2.3 Error Handling

Error handling deserves some mention. There are effectively four sources of error that are detected:

- Ran off the end of a table.
- Serial when token expected.

One complication occurs because in certain situations, what looks like an error, is not actually an error. The most important place where this occurs is when reading the macroblock address. It is legal in the syntaxes of MPEG, H.261 and JPEG for a Token to occur in place of the expected macroblock address. If this occurs in a legal manner, the Huffman error register is loaded with zero (meaning no error) but the Parser State Machine is still interrupted. The Parser State Machine's code must recognize this "no error" situation and respond accordingly. In this case, the "Huffman Error" event bit will not be set and the block will not stop processing.

Several situations must be dealt with. First, the Token occurs immediately with no preceding serial bits. In this case, a "Token when serial expected error" would occur. Instead, a "no error" error occurs in the way just described.

Second, the Token is preceded by a few serial bits. In this case, a decision is made. If all of the bits preceding the Token had the value one (remember that in H.261 and MPEG the coded data is inverted so these are zero bits in the coded data file) then no error occurs. If, however, any of them were zero, then they are not valid stuffing bits and, thus, an error has occurred and a "Token when serial expected" error does occur.

Third, the token is preceded by many bits. In this case, the same decision is made. If all sixteen bits are one, then they are treated as padding bits and a "no error" error occurs. If any of them had been zero, then "Ran off Huffman Table" error occurs.

Another place that a token may occur unexpectedly is in JPEG. When dealing with either Huffman tables or Quantizer tables, any number of tables may occur in the same Marker Segment. The Huffman Decoder does not know how many there are. Because of this fact, after each table is completed it reads another 4-bit FLC assuming it to be a new table number. If, however, a new marker segment starts, then a

2	Table[2]	or the number of bits to read for a FLC
1	Table[1]	
0	Table[0]	

Table B.2.2 Hufman Decoder Commands

B.2.2.4.1 Reading FLC

In this mode, Ignore Errors, Download, Alutab, Token, First Coeff, Special and VLC are all zero. Bypass will be set so that no Index to Data translation occurs.

The binary number in Table[3:0] indicates how many bits are to be read.

The numbers 0 to 12 are legal. The value zero does indeed read zero bits (as would be expected) and this instruction is, therefore, the Huffman Decoder NOP instruction. The values 13, 14 and 15 will not work and the value 15 is used when the Huffman State Machine is in control to denote the use of "SSSS" as the number of bits of FLC to read.

B.2.2.4.2 Reading VLC

In this mode, Ignore Errors, Download, Alutab, Token, First Coefficient and Special are zero and VLC is one. Bypass will usually be zero so that Index to Data translation occurs.

In this mode Token, First Coefficient and Special are all zero, VLC is one.

The binary number in Table[3:0] indicates which table to use as shown:

B.2.2.4.3 NOP Instructi n

As previously described, the action of reading a FLC of zero bits is used as a No Operation instruction. No data is read from the input ports (either Token or Serial) and the Huffman Decoder outputs a data value of zero along with the instruction word.

B.2.2.4.4 TCoefficient First Coefficient

The H.261 and MPEG TCoefficient Table has a special non-Huffman code that is used for the very first coefficient in the block. In order to decode a TCoefficient at the start of a block, the First Coefficient bit may be set along with a VLC instruction with table zero. One of the many effects of the First Coefficient bit is to enable this code to be decoded.

Note that in normal operation, it is unusual to issue a "simple" command to read a TCoefficient VLC. This is because control is usually handed to the Huffman Decoder by setting the Special Bit.

B.2.2.4.5 Reading Token Words

In order to read Token words, the Token bit should be set to one. The Special and First Coefficient bits should be zero. The VLC bit should also be set if the Table[0] bit is to work correctly.

In this mode, the bits Table[1] and Table[0] are used to modify the behavior of the Token reading as follows:

Bit	Meaning
Table[0]	Discard padding bits of serial data
Table[1]	Discard all serial data.

In the case of fixed length codes, the correct number of bits are read for decoding the vectors. If r_size is zero, a NOP instruction results.

In the case of Huffman codes, the generated table number has table[3] set to one so that the resulting number refers to one of the JPEG tables.

B.2.2.4.7 Special Instructions

All of the instructions (or modes of operation) described thus far are considered as "Simple" instructions. For each command that is received, the appropriate amount of input data (of either serial of token data) is read and the resulting data is output. If no error is detected, exactly one output will be generated per command.

In the present invention, special instructions have the characteristic that more than one output word may be generated for a single command. In order to accomplish this function, the Huffman Decoder's internal State Machine takes control and will issue itself instructions as required until it decides that the instruction which the Parser requested has been complete.

In all Special instructions, the first real instruction of the sequence that is to be executed is issued with the Special bit set to one. This means that all sequences must have a unique first instruction. The advantage of this scheme is that the first real instruction of the sequence is available without a look-up operation being required based upon the command received from the Parser.

There are four recognized special instructions:

- TCoefficient
- JPEG DC
- JPEG AC
- Token

The first of these reads H.261 and MPEG Transform coefficients, and the like, until the end-of-block symbol is read. If the block is a non-intra block, this command will read the entire block. In this case, the "First Coefficient" bit should be set so that the first

Tabl B.2.5 JPEG Tables

table[3:0]	Table nominated
10xx	JPEG DC Codes per bit
11xx	JPEG AC Codes per bit
00xx	JPEG DC Index to Data
01xx	JPEG AC Index to Data

As the above table shows, either the AC or DC tables can be loaded and table[3] determines whether it is the codes-per-bit table (in the Huffman decoder itself) or the Index to Data table that is loaded.

Once the table is nominated, data is downloaded into it by issuing a command to read the required number of FLC (always 8 bits) with the Download bits set (and the First Coeff bit zero). This causes the decoded data to be written into the nominated table. An address counter is maintained, the data is written at the current address and then the address counter is incremented. The address counter is reset to zero whenever a table is nominated.

When downloading the Index to Data tables, the data and addresses are monitored. Note that the address is the Huffman Index number while the data loaded into that address is the final decoded symbol. This information is used to automatically load the registers that hold the Huffman index number for symbols of interest. Accordingly, in a JPEG AC table, when the data has the value corresponding to ZRL is recognized, the current address is written into the register CED_H_KEY_ZRL_INDEX0 or CED_H_KEY_ZRL_INDEX1 as indicated by the table number.

Since decoded data is written into the codes-per-bit table one phase after it has been decoded, it is not possible to read data from the table during this phase.

provided by values obtained from the ALU register file (e.g., in the case of AC and DC table numbers and F-numbers). These values are stored in the auxiliary command storage, so that when that command is later reused the table number is that which has been stored. It is not recovered again from the ALU since, in general, the counters will have advanced in order to refer to the next block.

Since the choice of the next instruction that will be used depends upon the data that is being decoded, it is necessary for the decision to be made very late in a cycle. Accordingly, the general structure is one in which all of the possible instructions are prepared in parallel and multiplexing late in the cycle determines the actual instruction.

Note that in each case, in addition to determining the instruction that will be used by the Huffman Decoder in the next cycle, the state machine ROM also determines the instruction that will be attached to the current data as it passes to the Index to Data Unit and then onto the ALU. In exactly the same way, all three of these instructions are prepared in parallel and then a choice is made late in the cycle.

Again, there are three choices for this part of the instruction that correspond to the three choices for the next Huffman Decoder instruction above.

1)A constant instruction suitable for End of Block.

2)The Huffman State Machine. The Huffman State Machine may provide an arbitrary instruction for the Index to Data Unit.

3)The original instruction that was issued by the Parser to start the instruction.

B.2.2.5.1 EOB Comparator

The EOB comparator's output essentially forces selection of the constant instruction to be presented to the Index to Data Unit and will also cause the next Huffman Instruction nxtstate[4:0]

The address to use in the next cycle. This address may be modified.

statectl

Allows modification of the next state address. If zero, the state machine address is unmodified, otherwise the LSB of the address is replaced by the value of either of the two comparators as follows:

nxtstate[0]	
0	Replace Lsb by EOB match
1	Replace Lsb by ZRL match

Note: in any case, if the next Huffman Instruction is selected as "Re-run original command" the state machine will jump to location 0, 1, 2 or 3 as appropriate for the command.

eobct[1:0]

This controls the selection of the next Huffman instruction based upon the EOB comparator and extn bit as follows:

eobctl[1:0]	
00	No effect - see zrlctl[1:0]
01	Take new (Parser) command if EOB
10	Take new (Parser) command if extn low
11	Unconditional Demux Instruction

zrlct[1:0]

This controls the selection of the next Huffman instruction based upon the ZRL comparator. If the instruction. However, if the ZRL comparator matches then the zrltab[3:0] field is used in preference.

If it is not required that a different table number be used depending upon whether a ZRL match occurs, then both smtab[3:0] and zrltab[3:0] will have the same value. Note, however, that this can lead to strange simulation problems in Lsim. In the case of MPEG, there is no obvious requirement to load the register that indicate the Huffman index number for ZRL (a JPEG only construction). However, these are still selected and the output of the ZRL comparator becomes "unknown" despite the fact that both smtab[3:0] and zrltab[3:0] have the same value in all cases that the ZRL comparator may be "unknown" (so it does not matter which is selected) the next state still goes to "unknown".

zrltab[3:0]

This is the table number that will be used by the Huffman Decoder if the selected instruction is the state machine instruction and the ZRL comparator matches.

smvlc

This is the VLC bits used by the Huffman Decoder if the selected instruction is the state machine instruction.

aluzrl[1:0]

This field controls the selection of the instruction that is passed to the ALU. It will either be the command from the Parser State Machine (that was stored at the start of the instruction sequence) or the command from the state machine:

aluzrl[1:0]	
00	Always take the saved Parser State Machine Command
01	Always take the Huffman State Machine Command
10	Take the Huffman SM command if not EOB
11	Take the Huffman SM command if not ZRL

alueob

major blocks;

Table B.2.6 Huffman Modules

Module Name	Description
hddp	Huffman Decoder (Arithmetic) datapath
hdstdp	Huffman State Machine Datapath
hfitod	Index to Data Unit

The following description of the Huffman modules is accomplished by a global explanation of the various subsystem areas shown in greater detail in the drawings which are readily comprehended by one of ordinary skill in the art.

B.2.2.6.1 Description of "hd"

The logic for the two-wire interface control usually includes three ports controlled by the two-wire interface; data input, data output and the command. In addition, there are two "valid" wires from the input shifter; token_valid indicating that a Token is being presented on in_data[7:0] and serial_valid indicating that data is being presented on serial.

The most important signals generated are the enables that go to the latches. The most important being e1 which is the enable for the ph1 latches. The majority of ph0 latches are not enabled whilst two enables are provided for those that are; e0 associated with serial data and e0t associated with Token data.

In the present invention, the "done" signals (done, notdone and their ph0 variants done0 and notdone0) indicate when a primitive Huffman command is completed. In the case when a Huffman State Machine command is executed, "done" will be asserted at the completion of each primitive command that comprises the entire state-machine command.

signal is another (since no data should be read from the input shifter even though an output is produced to signal EOB) and lastly table download nomination is a third.

notfrczero (generated by a FLC of size zero, a NOP) ensures that the result is zero when a NOP instruction is used. Furthermore, invert indicates when the serial bits should be inverted before Huffman decoding (see section B.2.2.1.1). ring indicates when the transform coefficient ring should be applied (see section B.2.2.1.2).

Decoding is also accomplished regarding addressing the codes-per-bit ROMs. These are built out of the small data-path ROMs. The signals are duplicated (e.g., csha and csla) purely to get sufficient drive by separating the ROMs into two sections. The address can be taken either from the bit counter (bit[3:0]) or from the microprocessor interface address (key-addr[3:0]) depending upon UPI access to the block being selected.

Additional decoding is concerned with the UPI reading of registers such as those that hold the Huffman index values for the JPEG tables (EOB, ZRL etc.). Also included is a tristate driver control for these registers and the UPI reading of the codes per bit RAMs.

Arithmetic datapath decoding is also provided for certain important bit numbers. first_bit is used in connection with the Tcoeff first coefficient trick and bit_five is concerned with applying the ring in the Tcoeff table. Note the use of forceeob to simulate the action that the EOB comparator matches the decoded index value.

Regarding the extn bit, if a token is read from the input shifter, then the associated extn bit is read along with it. Otherwise, the last value of extn is preserved. This allows the testing of the extn bit by the microcode program at any time after a token has been read.

When zerodat is asserted, the upper four bits of the Huffman output data are forced to zero. Since these only have valid values when decoding fixed length codes, they are zeroed when decoding a VLC, a token or when a NOP conditions. These are ORed together in i_1190. In this case, i_1193, i-585 and i_584 constitute the three bit Huffman error register. Note i_1253 and i-1254 which disable the error in the cases when there is no "real" error (section B.2.2.3).

In addition, i_580 and i_579 along with the associated circuitry provide a simple state machine that controls the acceptance of the first command after an error is detected.

As previously indicated, control signals are delayed to match pipeline delays in the Index to Data Unit and the ALU.

Itod_bypass is the actual bypass signal passed to the Index to Data Unit. It is modified when the Huffman State Machine is in control to force bypass whenever a fixed length code is decoded.

Aluinstr[32] is the bit that causes the ALU to feedback (condition codes) to the Parser State Machine. Furthermore, it is important when the Huffman State Machine is in control that the signals are only asserted once (rather than each time one of the primitive commands completes).

Aluinstr[36] is the bit that allows the ALU to step the block counters (if other ALU instruction bits specify an increment too). This also must only be asserted once.

In addition, these bits must only be asserted for ALU instructions that output data to the Token Formatter. Otherwise, the counters may be incremented prior to the first output to the Token formatter causing an incorrect value of "cc" in a DATA token.

In the illustrated embodiment of the invention, either alunode[1] or alunode[0] will be low if the ALU will output to the Token Formatter.

Figure 118, similar to Figure 27, illustrates the Huffman State Machine datapath referred to as "hdstdp". There is also a UPI decode for reading the output of the Huffman State machine ROM.

Multiplexing is provided to deal with the case when the

The generation of escape_run is described in Section B.2.2.5.4.

Decoding also provides for the registers that hold the Huffman Index number for symbols such as ZRL and EOB. These registers can be loaded from the UPI or the datapath. The decoding in the center(es[4:0] and zs[3:0] is generating the select signals for the multiplexers that select which register or constant value to compare against the decode Huffman Index.

Regarding the control logic for the Huffman State Machine. Here the "instruction" bits from the Huffman State Machine ROM are combined with various conditions to determine what to do next and how to modify the instruction word for the ALU.

In the present invention, the signals notnew, notsm and notold are used on sheet 10 to control the operation of the Huffman Decoder command register. They are generated here in an obvious manner from the control bits in the state machine ROM (described in Section B.2.2.5.3) together with the output of the Huffman Index comparators (neobmatch and nzrlmatch).

Selection is also accomplished of the source for the instruction passed to the ALU. The actual multiplexing is performed in the Huffman State Machine datapath "hfstdp". Four control signals are generated.

In the case when the end-of-block has not been encountered, one of aluseldmx (selecting the Parser State Machine instruction) or aluselsm (selecting the Huffman state machine instruction) will be generated.

In the case when the end-of-block has not been encountered, one of aluseleobd (selecting the Parser State Machine instruction) or aluseleobs (selecting the Huffman State Machine instruction) will be generated. In addition the "outsrc" field of the ALU instruction is modified to force it to "zinput".

A register holds the nominated table number during table download. Decoding is provided for the codes-per-bit RAMs.

"Hdstmod" includes the Huffman State Machine ROM. Some bits of this instruction are used by the Huffman State Machine control logic. The remaining bits are used to replace that part of the ALU instruction word (from the Parser State Machine) that is not dealt with in "hdstdel".

"Hdstmod" is obvious and requires no explanation - there are only pipeline delay registers.

"Hdstdel" is also very simple and is handled by a ROM and multiplexers for modifying the ALU instruction. The remainder of the circuitry is concerned with UPI read access to half of the Huffman State Machine ROM outputs. Buffers are also used for the control signals.

B.2.3 The Token Formatter

The Huffman Decoder Token Formatter, in accordance with the present invention, sits at the end of the Huffman block. Its function, as its name suggests, is to format the data from the Huffman Decoder into the propriety Token structure. The input data is multiplexed with data in the Microinstruction word, under control of the Microinstruction word command field. The block has two operating modes; DATA_WORD, and DATA_TOKEN.

B.2.3.2.1 Data Word

In this mode, the top eight bits of the input are fed to the output. The bottom eight bits will be either the bottom eight bits of the input, the Token field of the Microinstruction word or a mixture of both, depending on the mask field. Mask represents the number of input bits in the mix, i.e.

out_data[16:8]=in_data[16:8]

out_data{7:0]=(Token[7:0]&(ff<<mask))indata[7:0]

When mask is set to 0 x 8 or greater, the output data will equal the input data. This mode is used to output words in non-DATA Tokens. With mask set to 0, out_data[7:0] will be the Token field of the Microinstruction word. This mode is used for outputting Token headers that contain no data. When Token headers do contain data, the number of data bits is given by the mask field.

If External Extn(Ee) is set, out_extn=in_extn, otherwise

out_extn=De.Bt and Eb are "don't care".

B.2.3.2.2 Data Token

This mode is used for formatting DATA Tokens and has two functions dependent on a signal, first_coefficient. At reset, first_coefficient is set. When the first data coefficient arrives along with a Microinstruction word that has cmd set to 1, out_data[16:2] is set to 0 x 1 and out_data[1:0] takes the value of the Bt field in the Microinstruction word. This is the header of a DATA Token. When this word has been accepted, the coefficient that accompanied the command is loaded into a register, RL and first_coefficient takes the value of Eb. When the next coefficient arrives, out_data[16:0] takes the previous coefficient, stored in RL. RL and first_coefficient are then updated. This ensures that when the end of the block is encountered and Eb is set, first_coefficient is set, ready for the next DATA Token, i.e.,

B.2.4 The Pars r State Machin

The Parser State Machine of the present invention is actually a very simple piece of circuitry. The complication lies in the programming of the microcode ROM which is discussed in Section B.2.5.

Essentially the machine consists of a register which holds the current address. This address is looked up in the microcode ROM to produce the microcode word. The address is also incremented in a simple incrementer and this incremented address is one of two possible addresses to be used for the next state. The other address is a field in the microcode ROM itself. Thus, each instruction is potentially a jump instruction and may jump to a location specified in the program. If the jump is not taken, control passes to the next location in the ROM.

A series sixteen condition code bits are provided. Any one of these conditions may be selected (by a field in the microcode ROM) and, in addition, it may be inverted (again a bit in the microcode ROM). The resulting signal selects between either the incremented address or the jump address in the microcode ROM. One of the conditions is hard-wired to evaluate as "False". If this condition is selected, no jump will occur. Alternatively, if this condition is selected and then inverted, the jump is always taken; an unconditional jump.

the feedback wire from the ALU becomes asserted. This wire, fb_valid, indicates that the condition codes currently being supplied by the ALU are valid in the sense that they reflect the data associated with the command that requested the wait for feedback.

The intended use of the feature, in accordance with the present invention, is in constructing conditional jump commands that decide the next state to jump to as a result of decoding (or processing) a particular piece of data. Without this facility it would be impossible to test any conditions depending upon data in the pipeline since the two-wire control means that the time at which a certain command reaches a given processing block (i.e., the ALU in this case) is uncertain.

Not all instructions are passed to the Huffman Decoder. Some instructions may be executed without the need for the data pipeline. These tend to be jump instructions. A bit in the microcode ROM selects whether or not the instruction will be presented to the Huffman Decoder. If not, there is no requirement that the Huffman Decoder accept the instruction and, therefore, execution can continue in these circumstances even if the pipeline is stalled.

B.2.4.2 Event Handling

There are two event bits located in the Parser State Machine. One is referred to as the Huffman event and the other is referred to as the Parser Event.

The Parser Event is the simplest of these. The "condition" being monitored by this event is simply a bit in the microcode ROM. Thus, an instruction may cause a Parser Event by setting this bit. Typically, the instruction that does this will write an appropriate constant into the rom_control register so that the interrupt service routine can determine the cause of the interrupt.

After servicing a Parser Event (or immediately if the event is masked out) control resumes at the point where it left off. If the instruction that caused the event has a

In this case, the coding_std register is not reset and control passes to the appropriate one of the first four locations.

The second four locations (0 x 004 to 0 x 007) are used when a Huffman interrupt takes place. Typically, a jump to the actual error handler is placed in each of these locations. Again, the choice of location is made as a result of the coding standard.

B.2.4.4 Tracing

As a diagnostic aid, a trace mechanism is implemented. This allows the microcode to be single-stepped. The bits CED_H_TRACE_EVENT and CED_H_TRACE_MASK in the register CED_H_TRACE control this. As their names suggest, they operate in a very similar fashion to the normal event bits. However, because of several differences (in particular no UPI interrupt is ever generated) they are not grouped with the other event bits.

The tracing mechanism is turned on when CED_H_TRACE_MASK is set to one. After each microcode instruction is read from the ROM, but before it is presented to the Huffman Decoder, a trace event occurs. In this case, CED_H_TRACE_EVENT becomes one. It must be polled because no interrupt will be generated. The entire microcode word is available in the registers CED_H_KEY_DMX_WORD_0 through CED_H_KEY_DMX_WORD_9. The instruction can be modified at this time if required. Writing a one to CED_H_TRACE_EVENT causes the instruction to be executed and clears CED_H_TRACE_EVENT. Shortly after this time, when the next microcode word to be executed has been read from the ROM, a new trace event will occur.

B.2.5 The Microcode

The microcode is programmed using an assembler "hpp" which is a very simple tool and much of the abstraction is achieved by using a macro preprocessor. A standard "C" preprocessor "cpp" may be used for this purpose.

The code is instructed as follows:

Ucode.u is the main file. First, this includes tokens.h

H_FLC_IE(NB) is like H_FLC, but the "ignore errors" bit is set.

H_TEST_VLC(TBL) is like H_VLC, but the bypass bit is set so that the Huffman Index is passed through the Index to Data Unit unmodified.

H_FWD_R and H_BWD_R read a FLC of the size indicated by the ALU registers r_fwd_r_size and r_bwd_r_size, respectively.

H_DCJ reads JPEG style DC coefficients, the table number from the ALU.

H_DCH reads a H.261 DC term.

H_TCOEFF and H_DCTCOEFF read transform coefficients. In H_DCTCOEFF, the first coeff bit is set and is for non-intra blocks, whilst H_TCOEFF is for intra blocks after the DC term has already been read.

H_NOMINATE(TBL) nominates a table for subsequent download.

H_DNL(NB) reads NB bits and downloads them into the nominated table.

<u>B.2.5.1.2 ALU Instructions</u>

There really are too many ALU instructions to explain them all in detail. The basic way in which the Mnemonics are constructed is discussed and this should make the instructions readable. Furthermore, these should readily be understandable to one of ordinary skill in the art.

Most of the ALU instructions are concerned with moving data from place to place and, therefore, a generic "load" instruction is used. In the Mnemonic, A_LDxy, it is understood that the contents of y are loaded into x., i.e., the destination is listed *first* and the source *second*:

the macroblock counter. These are fairly obvious and are described in comments in instr.u.

One anomaly is the use of a suffix "_O" to indicate that the result of the operation is output to the Token formatter in addition to the normal action. Thus LDFI_O(RA) stores the input data and also passes it to the token formatter. Alternatively, this could have been LDF_LDO_I(RA) if desired.

B.2.5.1.3 Token Formatter Instructions

This is the T_NOP "No-operation" instruction. This is really a misnomer as it is impossible to construct a no-operation instruction. However, this is used whenever the instruction is of no consequence because the ALU does not output to the Token Formatter.

T-TOK output a Token word.

T_DAT output a DATA Token word (used only with the Huffman State Machine instructions).

T-GENT8 generates a token word based on the 8 bits of constant field.

T_GENT8E like T_GENT8, but the extension bit is one.

T_OPD(NB) NB bits of data from the bottom NB bits of the output with the remainder of the bits coming from the constant field.

T_OPDE(NB) like T_OPD, but the extension bit is high.

T_OPD8 short-hand for T_OPD(8)

T_OPD8E short-hand for T_OPDE(8)

B.2.5.1.4 Parser State Machine Instructions

This instruction, D_NOP No-operation, i.e., the address increments as normal and the Parser State Machine does nothing special. The Remainder of the instruction is passed to the data pipeline. No waiting occurs.

D_WAIT is like D_NOP, but waits for feedback to occur.

The simple jump group. Mnemonics like D_JMP(ADDR) and D_JNX(ADDR) jump if the condition is met. The instruction is not output to the Huffman Decoder.

The external jump group. Mnemonics like D_XJMP(ADDR) and D_XJNX(ADDR). These are like their simple counterparts

D_EVENT causes generation of an event.

D_DFLT for construction of a default instruction. This causes an event and then jumps to a location with the label "dflt". This instruction should never be executed since they are used to fill a ROM so that a jump to an unused location is trapped.

D_ERROR causes an event and then jumps to a label "srch_dispatch" which is assumed to attempt recovery from the error.

Each of these blocks (except the output block) drives its output onto a bus running through the datapath, and these buses are, in turn, used as inputs to the multiplexing for block sources. For example, the adder output has it own datapath bus which is one of the possible inputs to the A register. Likewise, the A register has its own bus which forms one of the possible inputs to the adder. Only a subset of all possibilities exist in this respect, as specified in Section 7 on the microinstruction word.

In a single cycle, it is possible to execute either an add-based instruction or a sign-extend-based instruction. Furthermore, it is allowable to execute both of these in a single cycle provided that their operation is strictly parallel. In other words, add then sign extend or sign extend then add *sequences* are not allowed. The register file may be either read from or written to in a single cycle, but not both.

The output data has three fields:

- run - 6 bits
- sign - 1 bit
- level - 10 bits

If data is to be passed straight through the ALU, the least significant 11 bits of the input data register are latched into the sign and level fields.

It is possible to program limited multi-cycle operations of the ALU. In this regard, the number of cycles required is given by the contents of the register file location whose address is specified in the microinstruction, and the same operation is performed repeatedly while an iteration counter decrements to one. This facility is typically used to effect left shifts, using the adder to add the A register to itself and to store the result back in the A register.

B.3.3 The Adder/Subtractor Sub-Block

This is a 12-bit wide adder, with optional invert on its input2 and optional setting of the carry-in bit. Output is a 12 bit sum, and carry-out is not used. There are 7 modes is one, all bits remain unchanged. In both cases, sign takes the inverse of the size-th bit. This is used for obtaining the magnitude of AC difference values. For example:

```
data = 0000 1010 1010
size = 2
output = 0000 1010 1101, sign = 1

data = 0000 1010 1010
size = 1
output = 0000 1010 1010, sign = 0
```

In SGXMODE=DIFCOMP, all bits above (but not including) the size-th bit, take the inverse of the size-th bit, while all those below (and including) remain unchanged. Sign takes the inverse of the size-th bit. This is used for obtaining two's compliment values for DC difference values. For example:

```
data = 1010 1010 1010
size = 0
output = 1111 1111 1110, sign = 1
```

B.3.5 Condition Codes

There are two bytes (16 bits) of condition codes used by the Huffman block, certain bits of which are generated by the ALU/register file. These are the Sign condition code, the Zero condition code, the Extension condition code and a Change Detect bit. The last two of these codes are not really condition codes since they are not used by the Parser in the same way as the others.

The Sign, Zero and Extension condition codes are updated when the Parser issues an instruction to do so, and for each of these instructions the condition code valid signal is pulsed high once.

The Sign condition code is simply the sign extend sign output latched, while the Zero condition code is set to 1 if the input to the A register is zero. The Extension condition code is the input extension bit latched regardless of OUTSRC.

part of the address map for the UPI. Some multi-byte locations (denoted in the table by "O" for oversize) have a single ALU address, but multiple UPI addresses. Similarly, groups of registers which are indexed by the component count, CC (Indicated by I" in the table) are treated as a single location by the ALU. This eases microprogramming for initialization and resetting, and also for block-level operations.

All of the locations, except the dedicated ALU registers (UPI read only), are read/write, and all of the counters are reset to zero by a bit in the instruction word. The pattern code register has a right shift capability, its least significant bit forming the Pattern_Code condition bit. All registers in the hardwired macroblock structure are denoted in the table by "M", and those which are also counters (n-bit) are annotated with *Cn*.

In the present invention, certain locations have their contents hardwired to other parts of the Huffman sub-system-coding standard, two r-size locations, and a single location (2-bit word) for each of ac huff table and dc huff table to the Huffman Decoder.

Addresses in bold indicate that locations are accessible by both the ALU and the UPI, otherwise they have UPI access only. Groups of registers that are undirected through CC by the ALU can have a single ALU address specified in the instruction word and CC will select which physical location in the group to access. The ALU address may be that of any of the registers in the group, though conventionally, the address of the first should be used. This is also the case for multi-byte locations which should be accessed using the lowest address of the pair, although in practice, either address will suffice. Note that locations 2E and 2F are accessible in the top-level address map (denoted "T"), i.e., not only through the keyhole registers. These two locations are also reset to zero.

The register file is physically partitioned into four "banks" to improve access speed, but this does not affect

	Addr	Location			Addr	Location		
	00	A register 1		I	3E	c2		
	01	A register 0		I	3F	c3		
	02	run		I,O	40	dc pred_0 1		
	10	horiz pels 1		I,O	41	dc pred_0 0		
	11	horiz pels 0		I,O	42	dc pred_1 1		
	12	vert pels 1		I,O	43	dc pred_1 0		
	13	vert pels 0		I,O	44	dc pred_2 1		
	14	buff size 1		I,O	45	dc pred_2 0		
	15	buff size 0		I,O	46	dc pred_3 1		
	16	pel asp. ratio		I,O	47	dc pred_3 0		
	17	bit rate 2		O	50	prev mhf 1		
	18	bit rate 1		O	51	prev mhf 0		
	19	bit rate 0		O	52	prev mvf 1		
	1A	pic rate		O	53	prev mvf 0		
	1B	constrained		O	54	prev mhb 1		
	1C	picture type		O	55	prev mhb 0		
	1D	H261 picture type		O	56	prev mvb 1		
	1E	broken closed		O	57	prev mvb 0		
	1F	pred mode		M	60	mb horiz cnt1	C13	
	20	vbv delay 1		M	61	mb horiz cnt0	•	
	21	vbv delay 0		M	62	mb vert cnt1	C13	
	22	full pel fwd		M	63	mb vert cnt0	•	
	23	full pel bwd		M	64	horiz mb 1		
	24	horiz mb copy		M	65	horiz mb 0		
	25	pic number		M	66	vert mb 1		
	26	max h		M	67	vert mb 0		
	27	max v		M	68	restart count1	C16	
	28	-		M	69	restart count0	•	
	29	-		M	6A	restart gap1		
	2A	-		M	6B	restart gap0		
	2B	-		M	6C	horiz blk count	C2	
	2C	first group		M	6D	vert blk count	C2	
	2D	in picture		H,M	6E	comp id	C2	
T,R	2E	rom control		M	6F	max comp id		
T,R	2F	rom revision		H,R	70	coding std		
I,H	30	dc huff 0		M,H	71	pattern code	SR8	
I	31	dc huff 1		H	72	fwd r size		
I	32	dc huff 2		H	73	bwd r size		
I	33	dc huff 3						
I,H	34	ac huff 0						
I	35	ac huff 1						
I	36	ac huff 2		M,I	78	h0		

Table B.3.1 Table 1: Huffman Register File Address Map

	2B	-	
	2C	first scan	
	2D	in picture	
	2E	rom control	
	2F	rom revision	

Table B.3.2 JPEG Variations

H.261 Variations

	10	horiz pels 1	
	11	horiz pels 0	
	12	vert pels 1	
	13	vert pels 0	
	14	buff size 1	
	15	buff size 0	
	16	pel asp. ratio	
	17	bit rate 2	
	18	bit rate 1	
	19	bit rate 0	
	1A	pic rate	
	1B	constrained	
	1C	picture type	
	1D	H261 picture type	
	1E	broken closed	
	1F	pred mode	
	20	vbv delay 1	
	21	vbv delay 0	
	22	full pel fwd	
	23	full pel bwd	
	24	horiz mb copy	
	25	pic number	
	26	max h	
	27	max v	
	28	-	
	29	-	
	2A	-	
	2B	in gob	

Tabl B.3.3 H261 Variati ns

Field	Value	Description	Bits
OUTSRC	RSA6	run, sign, A register as 6 bits	0000
(specifies	ZZA	zero, zero, A register	0001
sources for	ZZA8	zero, zero, A register ls 8 bits	0010
run, sign and	ZZADDU4	zero, zero, adder o/p ms 4 bits	0011
level output)	ZINPUT	zero, input data	0100
	RSSGX	run, sign, sign extend o/p	0111
	RSADD	run, sign, adder o/p	1000
	RZADD	run, zero, adder o/p	1001
	RIZADD	input run, zero, adder output	
	ZSADD	zero, sign, adder o/p	1010
	ZZADD	zero, zero, adder o/p	1011
	NONE	no valid output - out_valid set to zero	11XX
REGADDR	00 - 7F	register file address for ALU access	7 bits
REGSRC	ADD	drive adder o/p onto register file i/p	0
	SGX	drive sign extend o/p onto register file i/p	1
REGMODE	READ	read from register file	0
	WRITE	write to register file	1
CNGDET	TEST	update change detect if REGMODE is WRITE	0
(change	HOLD	do not update change detect bit	1
detect)	CLEAR	reset change detect if REGMODE is READ	0

Tabl B.3.4 Table 2: Huffman ALU microinstructi n fi lds

(cond. codes)	HOLD	do not update condition codes	1
CNTMODE	NOCOUNT	do not increment counters	X00
(mbstructure	BCINCR	increment block counter and ripple	001
count mode)	CCINCR	force the component count to incr	010
	RESET	reset all counters in mb structure	011
	DISABLE	disable all counters	1XX
INSTMODE	MULTI	iterate current instr multi times	0
	SINGLE	single cycle instruction only	1

Table B.3.4 Table 2: Huffman ALU microinstruction fields

BASECB

·NUMBERCB - coded data buffer write pointer relative to READCB

·READTB - token data buffer read pointer relative to BASETB

·NUMBERTB - token data buffer write pointer relative to READTB

To calculate addresses:-

readaddr = (BASE + READ) mod LIMIT

writeaddr = (((READ + NUMBER) mod LENGTH) + BASE) mod LIMIT

The "mod LIMIT" term is used because a buffer may wrap around DRAM.

B.4.5 Block Description

In the present invention, and as shown in Figure 127, the Buffer Manager is composed of three top level modules connected in a ring which snooper monitors the DRAM interface connection. The modules are **bmprtize** (prioritize), **bminstr** (instruction), and **bmrecalc** (recalculate) are arranged in a ring of that order and **omsnoop** (snoopers) is arranged on the address outputs. The module,**Bmprtize**, deals with the REQ/ACK protocol, the FULL/EMPTY flags for the buffers and it maintains the state of each address, i.e., "is it a valid address?". From this information, it dictates to **bminstr** which (if any) address should be recalculated. It also operates the BUF_CSR (status) microprocessor register, showing FULL/EMPTY flags, and the buf_access microprocessor register, controlling microprocessor write access to the buffer manager registers.

The module, **Bminstr**, on being told by **bmprtize** to calculate an address, issues six instructions (one every two cycles) to control **bmrecalc** to calculating an address.

The module, **Bmrecalc**, recalculates the addresses under the instruction of **bminstr**. Running an instruction every two cycles, it contains all of the initialization and dynamic registers, and a simple ALU capable of addition, subtraction and modulus. It informs **Sbmprtize** of FULL/EMPTY

For read addresses:

Cycle	Operation	BusA	BusB	Result	Meaning of result's sign
0-1	ADD	READ	BASE		
2-3	MOD	Accum	LIMIT	Address	
4-5	ADD	READ	"1"		
6-7	MOD	Accum	LENGTH	READ	
8-9	SUB	NUMBER	"1"	NUMBER	
10-11	MOD	"0"	Accum		SET_EMPTY (NUMBER >= 0)

Table B.4.1 Read address calculation

For write addresses:

Cycle	Operation	BusA	BusB	Result	Meaning of result's sign
0-1	ADD	NUMBER	READ		
2-3	MOD	Accum	LIMIT		
4-5	ADD	Accum	BASE		
6-7	MOD	Accum	LIMIT	Address	
8-9	ADD	NUMBER	"1"	NUMBER	
10-11	MOD	Accum	LENGTH		SET_FULL (NUMBER >= LENGTH)

Table B.4.2 F r write addr ss calculati ns

the FIFO pointers.

B.4.7 Registers

To gain microprocessor write access to the initialization registers, a one should be written to buf_access, and access will be given when buf_access reads back one. Conversely, to give up microprocessor write access, zero should be written to buf_access. Access will be given when buf_access reads back zero. Note that buf_access is reset to one.

The dynamic and initialization registers of the present invention may be read at any time, however, to ensure that the dynamic registers are not changing the microprocessor, write access must be gained.

It is intended that the initialization registers be written to only once. Re-writing them may cause the buffers to operate incorrectly. However, it is envisioned to increase the buffer length on-the-fly and to have the buffer manager use the new length when appropriate.

No check is ever made to see that the values in the initialization registers are sensible, e.g., that the buffers do not overlap. This is the user's responsibility.

Keyhole Register Name	Usage	Key hole Address
CED_BUF_CB_NUMBER_2	xxxxxxDD	0x0d
CED_BUF_CB_NUMBER_1	DDDDDDDD	0x0e
CED_BUF_CB_NUMBER_0	DDDDDDDD	0x0f
CED_BUF_TB_BASE_3	xxxxxxxx	0x10
CED_BUF_TB_BASE_2	xxxxxxDD	0x11
CED_BUF_TB_BASE_1	DDDDDDDD	0x12
CED_BUF_TB_BASE_0	DDDDDDDD	0x13
CED_BUF_TB_LENGTH_3	xxxxxxxx	0x14
CED_BUF_TB_LENGTH_2	xxxxxxDD	0x15
CED_BUF_TB_LENGTH_1	DDDDDDDD	0x16
CED_BUF_TB_LENGTH_0	DDDDDDDD	0x17
CED_BUF_TB_READ_3	xxxxxxxx	0x18
CED_BUF_TB_READ_2	xxxxxxDD	0x19
CED_BUF_TB_READ_1	DDDDDDDD	0x1a
CED_BUF_TB_READ_0	DDDDDDDD	0x1b
CED_BUF_TB_NUMBER_3	xxxxxxxx	0x1c
CED_BUF_TB_NUMBER_2	xxxxxxDD	0x1d
CED_BUF_TB_NUMBER_1	DDDDDDDD	0x1e
CED_BUF_TB_NUMBER_0	DDDDDDDD	0x1f
CED_BUF_LIMIT_3	xxxxxxxx	0x20
CED_BUF_LIMIT_2	xxxxxxDD	0x21
CED_BUF_LIMIT_1	DDDDDDDD	0x22
CED_BUF_LIMIT_0	DDDDDDDD	0x23
CED_BUF_CSR	xxxxDDDD	0x24

Table B.4.4 Registers in buffer manager keyhole

B.4.8 Verification

Verification was conducted in Lsim with small FIFO's onto a dummy DRAM interface, and in C-code as part of the top level chip simulation.

SECTION B.5 Inverse Modeler

B.5.1 Introduction

This document describes the purpose, actions and implementation of the Inverse Modeller (**imodel**) and the Token Formatter (**hsppk**), in accordance with the present invention.

Note: **hsppk** is a hierarchically part of the Huffman Decoder, but functionally part of the Inverse Modeller. It is, therefore, better discussed in this section.

B.5.2 Overview

The Token buffer, which is between the **imodel** and **hsppk**, can contain a great deal of data, all in off-chip DRAM. To ensure that efficient use is made of this memory, the data must be in a 16 bit format.. The Formatter "packs" the data from the Huffman Decoder into this format for the Token buffer. Subsequently, the Inverse Modeler "unpacks" data from the Token buffer format.

However, the Inverse Modeller's main function is the expanding out of "run/level" codes into a run of zero data followed by a level. Additionally, the Inverse Modeller ensures that DATA tokens have at least 64 coefficients and it provides a "gate" for stopping streams which have not met their start-up criteria.

B.5.3 Interfaces

B.5.3.1 Hsppk

In the present invention, **Hsppk** has the Huffman Decoder as input and the Token buffer as output. Both interfaces are of the two-wire type, the input being a 17 bit token port, the output being 16 bit "packed data", plus a FLUSH signal. In addition, Hsppk is clocked from the Huffman clock generator and, thus, connected to the Huffman scan chain.

B.5.3.2 Imodel

Imodel has the Token buffer start-up output gate logic (**bsogl**) as inputs and the Inverse Quantizer as output. Input from the Token buffer is 16 bit "packed data", plus block_end signal, from the **bsogl** is one wirestream_enable.

unconditionally used.

B.5.4.1.2 Packing

After splitting, all data words are 12 bits wide. Every four 12 bit words are "packed" into three 16 bit words:

Input words	Output words
000000000000	0000000000001111
111111111111	1111111122222222
222222222222	2222333333333333
333333333333	

Table B.5.1 Packing method

B.5.4.1.3 Flushing of the buffer

The DRAM interface of the present invention collects a block, 32 sixteen bit "packed" words, before writing them to the buffer. This implies that data can get stuck in the DRAM interface at the end of a stream, if the block is only partially complete. Therefore a flushing mechanism is required. Accordingly, **.Hsppk** signals the DRAM interface to write it current partially complete block unconditionally.

B.5.4.2.1 Imup (UnPacker)

Imup performs three functions:

4)Unpacking data from its sixteen bit format into 12 bit words.

Input words	Output words
0000000000001111	000000000000
1111111122222222	111111111111
2222333333333333	222222222222
	333333333333

Table B.5.2 Unpacking method

output.

B.5.5.1.2 Packing

The packing procedure cycles every four valid data inputs. The sixteen bit word output is formed from the last valid word, which is held, and the succeeding word. If this is not valid, then the output is not valid. The procedure is:

	Held Word	Succeeding Word	Packed Word	
valid cycle 0	xxxxxxxxxxxx	000000000000	xxxxxxxxxxxxxxxx	don't output
valid cycle 1	000000000000	111111111111	0000000000001111	output
valid cycle 2	111111111111	222222222222	1111111122222222	output
valid cycle 3	222222222222	333333333333	2222333333333333	output

Table B.5.3 Packing procedure

Where x indicates undefined bits.

During valid cycle 0, no word is output because it is not valid.

The valid cycle number is maintained by a ring counter. It is incremented by valid data from the splitter and an accepted output.

When a FLUSH (or picture_end) token is received and the token itself is ready to output, a flush signal is also output to the DRAM interface to reset the valid cycle to zero. If a FLUSH token arrives on anything but cycle 3, the flush signal must be delayed a valid cycle to ensure the token itself it output.

B.5.5.2 Imodel

B5.5.2.1 Imup (Unpacker)

As with the packer, the last valid input is stored, and combined with the next input, allows unpacking.

If the token ends before there are 64 coefficients, zero coefficients are inserted at the end of the token to complete it to 64 coefficients. For example, unextended data headers have 64 zero coefficients inserted after them. DATA tokens with 64 or more coefficients are not affected by **impad**.

SECTION B.6 Buffer Start-up

B.6.1 Introduction

This section describes the method and implementation of the buffer start-up in accordance with the present invention.

B.6.2 Overview

To ensure that a stream of pictures can be displayed smoothly and continuously a certain amount of data must be gathered before decoding can start.. This is called the start-up condition. The coding standard specifies a VBV delay which can be translated, approximately, into the amount of data needed to be gathered. It is the purpose of the "Buffer Start-up" to ensure that every stream fulfills its start-up condition before its data progresses from the token buffer, allowing decoding. It is held in the buffers by a notional gate (the output gate) at the output of the token buffer (i.e., in the Inverse Modeler). This gate will only be open for the stream once its start-up condition has been met.

B.6.3 Interfaces

Bscntbit (Buffer Start-up bit counter) is in the datapath, and communicates by two-wire interfaces, and is connected to the microprocessor. It also branches with a two-wire interface to **bsogl** (Buffer Start-up Output Gate Logic). **Bsogl** via a two-wire interface controls **imup** (Inverse Modeler UnPacker), which implements the output gate.

B.6.4 Block Structure

As shown in Figure 130, **Bscntbit** lies in the datapath between the Start Code Detector and the coded data buffer. This single cycle block counts the valid words of data leaving the block and compares this number with the start-up condition (or target) which will be loaded from the microprocessor. When the target is met, **bsogl** is informed. Data is unaffected by **bscntbit**.

Bsogl lies between **bscntbit** and **imup** (in the inverse modeler). In effect, it is a queue of indicators that streams have met their targets. The queue is moved along

bsbitcnt, an abs_flush_event is generated. If the stream ends before the target is met, an additional event is also generated (bs_flush_before_target_met_event). When any of these events occur, the block is stalled. This allows the user to recommence the search for the next stream's target or in the case of a bs_flush_before_target_met_event event either:

1) write a target of zero which will force a target_met or

2) note that target was not met and allow the next stream to proceed until this combined with the last stream reaches the target. The target for this next stream can should adjusted accordingly.

B.6.5.2 BSOGL (buffer start-up output gate logic)

As previously described, **Bsogl** is a queue of indicators that a stream has met its target. The queue type is set by ced_bs_queue (internal(0) or external(1)). This is a reset to select an internal queue. The depth of the queue determines the maximum number of satisfied streams that can be in the coded data buffer, Huffman, and token buffer. When this number is reached (i.e. the queue is full) **bsogl** will force the datapath to stall at **bsbitcnt**.

Using an internal queue requires no action from the microprocessor. However, if it is necessary to increase the depth of the queue, an external queue can be set (by setting ced_bs_access to gain access to ced_bs_queue which should be set, target_met_event and stream_end_event enabled and access relinquished).

The external queue (a count maintained by the microprocessor) is inserted into the internal queue. The external queue is maintained by two events. target_met_event and stream_end_event. These can simply be referred to as service_queue_input and service_queue_output respectively] and a register ced_bs_enable_nxt_stream. In effect, target_met_event is the up stream end of the internal queue supplying the queue. Similarly, ced_bs_enable_nxt_stream is the down stream end of the

The queue type can be changed from internal to external at any time (by the means described above), but they can only be changed external to internal when the external queue is empty (from above "queue==0"), by setting ced_bs_access to gain access to ced_bs_queue which should be reset, target_met_event and stream_end_event masked, and access relinquished.

On the other hand, disable checking of stream start-up conditions, set ced_bs_queue (external), mask target_met_event and stream_end_event and set ced_bs_enable_nxt_stream. In this way, all streams will always be enabled.

where

- D is a register bit
- x is a non-existent register bit
- r is a reserved register bit
- to gain access to these registers ced_bs_access must be set to one and polled until it reads back one, unless in an interrupt service routine. Access is given up by setting ced_bs_access to zero.

Blue color difference data (Cr and Cb, r spectively).

The following section describes the operation of a DRAM interface in accordance with the present invention, which has one write swing buffer and one read swing buffer, which is essentially the same as the operation of the Spatial Decoder DRAM Interface. This is illustrated in Figure 131, "DRAM Interface,".

B.7.2 A Generic DRAM Interface

Referring to Figure 131, the interfaces to the address generator 420 and to the blocks which supply and take the data are all two wire interfaces. The address generator 420 may either generate addresses as the result of receiving control tokens, or it may merely generate a fixed sequence of addresses. The DRAM interface 421 treats the two wire interfaces associated with the address generator in a special way. Instead of keeping the accept line high when it is ready to receive an address, it waits for the address generator to supply a valid address, processes that address and then sets the accept line high for one clock period. Thus, it implements a request/acknowledge (REQ/ACK) protocol.

A unique feature of the DRAM Interface is its ability to communicate with the address generator and the blocks which provide or accept the data completely independent of the other. For example, the address generator may generate an address associated with the data in the write swing buffer, but no action will be taken until the write swing buffer signals that there is a block of data which is ready to be written to the external DRAM 422. However, no action is taken until an address is supplied on the appropriate bus from the address generator. Further, once one of the RAMs in the write swing buffer has been filled with data, the other may be completely filled and "swung" to the DRAM Interface side before the data input is stalled (the two-wire interface accept signal set low).

In understanding the operation of the DRAM Interface of the present invention, it is important to note that in a be accepted by the swing buffer continually, otherwise when RAM2 is filled the swing buffer will set its accept signal low until RAM1 has been "swung" back for use by the input side.

6)This process is repeated ad infinitum.

The operation of a read swing buffer is similar, but with input and output data busses reversed.

B.7.2.2 Addressing of External DRAM and Swing Buffers

The DRAM Interface is designed to maximize the available memory bandwidth. Consequently, it is arranged so that each 8x8 block of data is stored in the same DRAM page. In this way full use can be made of DRAM fast page access modes, where one row address is supplied followed by many column addresses. In addition, a facility is provided to allow the data bus to the external DRAM to be 8, 16 or 32 bits wide, so that the amount of DRAM used can be matched to the size and bandwidth requirements of the particular application.

In this example (which is exactly how the DRAM Interface on the Spatial Decoder works), the address generator provides the DRAM Interface with block addresses for each of the read and write swing buffers. This address is used as the row address for the DRAM. The six bits of column address are supplied by the DRAM Interface itself, and these bits are also used as the address for the swing buffer RAM. The data bus to the swing buffers is 32 bits wide, so if the bus width to the external DRAM is less than 32 bits, two or four external DRAM accesses must be made before the next word is read from a write swing buffer or the next word is written to a read swing buffer (read and write refer to the direction of transfer relative to the external DRAM).

The situation is more complex in the cases of the Temporal Decoder and the Video Formatter. These are covered separately below.

B.7.3 DRAM Int rfac Timing

In the present invention, the DRAM Interface Timing block

SECTION B.8 Inverse Quantizer

B.8.1 Introduction

This document describes the purpose, actions and implementation of the inverse quantizer, (iq) in accordance with the present invention.

B.8.2 Overview

The inverse quantizer reconstructs coefficients from quantized coefficients, quantization weights and step sizes, all of which are transmitted within the datastream.

B.8.3 Interfaces

The iq lies between the inverse modeler and the inverse DCT in the datapath and is connected to a microprocessor. Datapath connections are via two-wire interfaces. Input data is 10 bits wide, output is 11 bits wide.

B.8.4 Mathematics of Inverse Quantization

B.8.4.1 H261 Equations

For blocks coded in intra mode:

$$C'_i = 8Q_i \qquad i = 0$$

$$\left.\begin{array}{ll} C''_i = \text{iq\_quant\_scale}\,[2Q_i + sign\,(Q_i)\,] & \\ C'_i = C''_i - sign\left(C''_i\right) & C''_i = \text{even} \\ C'_i = C''_i & C''_i = \text{odd} \end{array}\right\} 0 < i < 64$$

$$C_i = min(max(C''_i, -2048), 2047)$$

For all other coded blocks:

$$\left.\begin{array}{ll} C''_i = \text{iq\_quant\_scale}\,[2Q_i + sign\,(Q_i)\,] & \\ C'_i = C''_i - sign\left(C''_i\right) & C''_i = \text{even} \\ C'_i = C''_i & C''_i = \text{odd} \end{array}\right\} \quad 0 \le i < 64$$

$$C_i = min(max(C''_i, -2048), 2047)$$

B.8.4.5 All other tokens

All tokens except DATA Tokens must pass through the iq unquantized

Where:

$$sign(a) = \begin{cases} -1 & a < 0 \\ 0 & a = 0 \\ 1 & a > 0 \end{cases}$$

$$max(a,b) = \begin{cases} a & a > b \\ b & a \le b \end{cases}$$

$$min(a,b) = \begin{cases} a & a \le b \\ b & a > b \end{cases}$$

Floor(a) returns an integer such that:

$$\begin{array}{ll} (a-1) < floor(a) \le a & a \ge 0 \\ a \le floor(a) < (a+1) & a \le 0 \end{array}$$

Q_i are the quantized coefficients.

C_i are the reconstructed coefficients

$W_{i,j}$ are the values in the quantisation table matrices

i is the coefficient index along the zig-zag

j is the quantisation table matrix number (0 <= j <=3)

B.8.4.6 Multiple Standards combined

It can be shown that all the above standards and their variations (also control data which must be unchanged by the iq) can be mapped on to single equation:

$$OUTPUT = \frac{(2INPUT + k)(xy)}{16}$$

With the additional post inverse quantisation functions of :

•Add 1024

•Convert from sign magnitude to 2's complement representation.

•Round all even numbers to the nearest odd number towards zero.

•Saturate result to +2047 or -2048.

The variables k, x and y for each variation of the standards and which functions they use is shown in Table B.8.1.

Figure 134.

B.8.6 Block Implementation

B.8.6.1 Iqca

In the invention, **iqca** is a state machine used to decode tokens into control signals for **igram** and the register in **iqcb**. The state machine is better described as a stat machine for each token since it is reset by each new token. For example:

The code for the QUANT_SCALE (see B.8.7.4, "QUANT_SCALE") and QUANT_TABLE (see B.8.7.6, "QUANT_TABLE") are as follows:

```
if (tokenheader == QUANT_SCALE)
{
  sprintf(preport, "QUANT_SCALE");
  reg_addr = ADDR_IQ_QUANT_SCALE;
  rnotw = WRITE;
  enable = 1;
}

if (tokenheader == QUANT_TABLE) /*QUANT_TABLE token */
switch (substate)
{
  case 0: /* quantisation table header */
    sprintf(preport, "QUANT_TABLE_%s_s0",
      (headerextn ? "(full)" : "(empty)"));
    nextsubstate = 1;
    insertnext = (headerextn ? 0 : 1);
    reg_addr = ADDR_IQ_COMPONENT;
    rnotw = WRITE;
    enable = 1;
```

Where a substate is a state within a token, QUANT_SCALE has, for example, only one substate. However, the QUANT_TABLE has two, one being the header, the second the token body.

The state machine is implemented as a PLA. Unrecognized tokens cause no wordline to rise and the PLA to output default (harmless) controls.

Additionally, **iqca** supplies addresses to **igram** from BodyWord counter and inserts words into the stream, for example in an unextended QUANT_TABLE (see B.8.7.4). This is achieved by stalling the input while maintaining the output valid. The words can be filled with the correct data in succeeding blocks (**iqcb** or **iqarith**).

iqca is a single cycle in the datapath controlled by two-wire interfaces.

B.8.6.2 iqcb

In the invention, **iqcb** holds the **iq** status registers. Under the control of **iqca** it loads or unloads these from/to the datapath.

The status registers are decoded (see Table B.8.1) into control wires for **iqarith**; to control the XY multiplier terms and the post quantization functions.

The sign bit of the datapath is separated here and sent to the post quantization functions. Also, zero valued words on the datapath are detected here. The arithmetic is then ignored and zero muxed onto the datapath. This is the easiest way to comply with the "zero in; zero out" spec of the **iq**.

The status registers are accessible from the microprocessor only when the register iq_access has been set to one and reads back one. In this situation, **iqcb** has halted the datapath, thus ensuring the registers have a stable value and no data is corrupted in the datapath.

Iqcb is a single cycle in the datapath controlled by two wire interfaces.

B.8.6.3 Iqram

Iqram must hold up to four quantization table matrices

B.8.6.4.3 Post quantization functions

The post quantization functions are

- Add 1024
- Convert from sign magnitude to 2's complement representation.
- Round all even numbers to the nearest odd number towards zero.
- Saturate result to +2047 or -2048.
- Set output to zero (see B.8.6.2)

The first three functions are implemented on a 12 bit adder (pipelined over the second and third cycles). From this, it can be seen what each function requires and these are then combined onto the single adder.

Function	if datapath > 0	if datapath > 0
Convert to 2's complement	nothing	invert add one
Round all even numbers	subtract one	add one

Function	if datapath > 0	if datapath > 0
Add 1024	add 1024	add 1024

Table B.8.2 Post quantization adder functions

As will be appreciated by one of ordinary skill in the art, care should be taken when reprogramming these functions as they are very interdependent when combined.

The saturate values, zero and zero+1024 are muxed onto the datapath at the end of the third cycle.

B.8.7 Inverse Quantizer Tokens

The following notes define the behavior of the Inverse Quantizer for each Token tp which it responds. In all cases, the Tokens are also transported to the output of the in twelve bits in a twos complement format (eleven bits plus a sign bit). The DATA Token at the output will have the same numb r of Token Extension words as it had at the input of the Inverse Quantizer.

B.8.7.6 QUANT_TABLE

This Token may be used to load a new quantization table or to read out an existing table. Typically, in the Inverse Quantizer, the Token will be used to load a new table which has been decoded from the bit stream. The action of reading out an existing table is useful in the forward quantizer of an encoder if that table is to be encoded into the bit stream.

The Token Head contains two bits identifying the table number that is to be used. These are placed in iq_component[1:0]. Note that this register now contains a "table number" not a color component.

If the extension bit of the Token Head is one, the Inverse Quantizer expects there to be exactly sixty four extension Token Words. Each one is interpreted as a quantization table value and placed in a successive location of the appropriate table, starting at location zero. The ninth bit of each extension Token word is ignored. The Token is also passed to the output of the Inverse Quantizer, unmodified, in the normal way.

If the extension bit of the Token Head is zero, then the Inverse Quantizer will read out successive locations of the appropriate table starting at location zero. Each location becomes an extension Token word (the ninth bit will be zero). At the end of this operation, the Token will contain exactly sixty four extension Token words.

The operation of the Inverse Quantizer in response to this token is undefined for all numbers of extension words except zero and sixty four.

B.8.7.7 JPEG_TABLE_SELECT

This token is used to load or unload translations of color components to table numbers to/from

B.8.8 Microprocessor Registers

B.8.8.1 iq_access

To gain microprocessor access to any of the iq registers, iq_access must be set to one and polled until it reads back one (see B.8.6.2). Failure to do this will result in the registers being read still being controlled by the datapath and, therefore, not being stable. In the case of the **iqram**, the accesses are locked out, reading back zeros.

Writing zero to iq_access relinquishes control back to the datapath.

B.8.8.2 Iq_coding_standard[1:0]

This register holds the coding standard that is being implemented by the Inverse Quantizer.

iq_coding_standard	Coding Standard
0	H.261
1	JPEG
2	MPEG
3	XXX

Table B.8.4 Coding standard values

This register is loaded by the CODING_STANDARD Token.

Although this is a two bit register, at present eight bits are allocated in the memory map and future implementations can deal with more than the above standards.

The JPEG_TABLE_SELECT Token causes this register be loaded with a color component. It is then used as an index into iq_ipeg_indirection[7:0] which is accessed by the tokens body.

The QUANT_SCALE Token causes this register to be loaded with the QTM number. This table is then either loaded from the Token (if the extended form of the Token is used) or read out from the table to form a properly extended Token.

B.8.8.7 iq_prediction_mode[1:0]

This two bit register holds the prediction mode that will be used for subsequent blocks. The only use that the Inverse Quantizer makes of this information is to decide whether or not intra coding is being used. If both bits of the register are zero, then subsequent blocks are intra coded.

This register is loaded by the PREDICTION_MODE Token. This register is reset to zero by the SEQUENCE_START Token.

Iq_prediction_mode[1:0] has no effect on the operation in JPEG and JPEG variation modes.

B.8.8.8 Iq_ipeg_indirection[7:0]

Iq_ipeg_indirection is used as a lookup table to translate color components into the QTM number. Accordingly, iq_component is used as an index to iq_ipeg_indirection as shown in Table B.8.3.

This register location is written to directly by the JPEG_TABLE_SELECT Token if the extended form of the Token is used.

This register location is read directly by the JPEG_TABLE_SELECT Token if the non-extended form of the Token is used.

B.8.8.9 Iq_quant_table[3:0][63:0][7:0]

There are four quantization tables, each with 64 locations. Each location is an eight bit value. The value zero should not be used in any location.

These registers are implemented as a RAM described in B.8.6.3, "Igram".

These tables may be loaded using the QUANT_TABLE

B.8.10 T st

Test coverage to the Inverse Quantizer at the input is through the Inverse Modeler's output snooper, and at the output through the Inverse Quantizer's own snooper. Logic is covered by the Inverse Quantizer's own scan chain.

Access can be gained to **igram** without reference to iq_access if the ramtest signal is asserted.

compression, although run-length coding is generally not a lossy process.

The IDCT block (which actually includes an Inverse Zig-Zag RAM, or IZZ, and an IDCT) takes frequency data, which is sorted, and transforms it into spatial data. This inverse sorting process is the function of IZZ.

The picture decompression system, of which the IDCT block forms a part, specifies the pixels as integers. This means that the IDCT block must take, and yield, integer values. However, since the IDCT function is not integer based, the internal number representation uses fractional parts to maintain internal accuracy. Full floating-point arithmetic is preferable, but the implementation described herein uses fixed-point arithmetic. There is some loss of accuracy using fixed-point arithmetic, but the accuracy of this implementation exceeds the accuracy specified by H.261 and the IEEE.

B.9.3 Design Objectives

The main design objective, in accordance with the present invention, was to design a functionally correct IDCT block which uses a minimum silicon area. The design was also required to run with a clock speed of 30MHz under the specified operating conditions, but it was considered that the design should also be adaptable for the future. Higher clock rates will be needed in the future, and the architecture of the design allows for this wherever possible.

B.9.4 IDCT Interfaces Description

The IDCT block has the following interfaces.

- a 12-bit wide Token data input port
- a 9-bit wide Token data output port
- a microprocessor interface port
- a system services input port
- a test interface
- resynchronizing signals

Both the Token data ports are the standard Two-Wire Interface type previously described. The widths uses a one-dimensional IDCT algorithm designed specifically for mapping onto hardware; the algorithm is not appropriate for software models. The one-dimensional algorithm is applied successively to obtain a two-dimensional result. The mathematical definition of the two-dimensional DCT for an N by N block of pixels is as follows:

EQ 10. **forward DCT**

$$Y(j,k) = \frac{2}{N} c(j)\, c(k) \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} X(m,n) \cos\left[\frac{(2m+1)\,j\pi}{2N}\right] \cos\left[\frac{(2n+1)\,k\pi}{2N}\right]$$

EQ 11. **inverse DCT**

$$X(m,n) = \frac{2}{N} \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} c(j)\, c(k)\, Y(j,k) \cos\left[\frac{(2m+1)\,j\pi}{2N}\right] \cos\left[\frac{(2n+1)\,k\pi}{2N}\right]$$

Where

$$j,k = 0, 1, \ldots, N-1$$

$$c(j)\, c(k) = \begin{cases} \frac{1}{\sqrt{2}} & j,k = 0 \\ 1 & otherwise \end{cases}$$

dimensional answer will be scaled by 2. This can then be easily corrected in the final saturation and rounding stage by shifting.

The algorithm shown was coded in double precision floating-point C and the results of this compared with a reference IDCT (using straightforward matrix multiplication). A further stage was then used to code a bit-accurate integer version of the algorithm in C (no timing information was included) which could be used to verify the performance and accuracy of the algorithm as it would be implemented on silicon. The allowable inaccuracies of the transform are specified in the H.261 standard and this method was used to exercise the bit-accurate model and measure the delivered accuracy.

Figure 137 shows the overall IDCT Architecture in a way that illustrates the commonality between the upper and lower sections and which also shows the points at which intermediate results need to be stored. The circuit is time multiplexed to allow the upper and lower sections to be calculated separately.

B.9.7 The IDCT Transform Architecture

As described previously, the IDCT algorithm is optimized for an efficient architecture. The key features of the resulting architecture are as follows:

- significant re-use of the costly arithmetic operations
- small number of multipliers, all being constant coefficient rather than general purpose (reduces multiplier size and removes need for separate coefficient store)
- small number of latches, no more than required for pipelining the architecture
- operations are arranged so that only a single resolving operation is required per pipeline stage
- can arrange to generate results in natural order
- no complex crossbar switching or significant multiplexing (both costly in a final implementation)

control of the fixed-point position.

·Making use of statistical definition of the accuracy requirement in order to achieve accuracy by selective manipulation of arithmetic operations (rather than increasing accuracy by simply increasing the word width of the entire transform)

The straightforward way to design a transform would involve a simple fixed-point implementation with a fixed word-width made large enough to achieve accuracy. Unfortunately, this approach results in much larger word widths and, therefore, a larger transform. The approach used in the present invention allows the fixed point position to vary throughout the transform in a manner that makes the maximum use of the available dynamic range for any particular intermediate value, achieving the maximum possible accuracy.

Because the allowable results are specified statistically, selective adjustments can be made to any intermediate value truncation operation in order to improve overall accuracy. The adjustments chosen are simple manipulations of LSB calculations, which have little or no cost. The alternative to this technique is to increase the word width, involving significant cost. The adjustments effectively "weight" final results in a given direction, if it is found that previously, these results tend in the opposite direction. By adjusting the fractional parts of results, we are effectively shifting the overall average of these results.

B.9.8 IDCT Block Diagram Description

The block diagram of the IDCT shows all the blocks that are relevant to the processing of the Token Stream. This diagram, Figure 138, does not show details of clocking, test and microprocessor access and the event mechanism. Snooper blocks, used to provide test access, are not shown in th diagram.

B.9.8.1 DATA Error Checker

The first block is the DATA error checker and corrector,

B.9.8.3 Input F rmatt r

The next block in Figure 138 is the input formatter 442, "ip_fmt", which formats DATA input for the first dimension of the IDCT transform. This block has a 12-bit wide Token Stream input and 22-bit wide token Stream output. DATA Tokens are shifted left so as to move the integer part to the correct significance in the IDCT transform standard 22-bit wide word, the fractional part being set to 0. This means that there are 10 bits of fraction at this point. All other Tokens are unshifted and the extra unused bits are simply set to 0.

B.9.8.4 1-Dimensional Transform - 1st Dimension

The next block shown in Figure 138 is the first single dimension IDCT transform block 443,"oned". This inputs and outputs 22-bit wide token Streams and, as usual, the stream is parsed and DATA Tokens are recognized. All other tokens are passed through unaltered. The DATA Tokens pass through a pipelined datapath that performs an implementation of a single dimension of an 8-by-8 Inverse Discrete Cosine Transform. At the output of the first dimension, there are 7 bits of fraction in the data word. All other Tokens run through a merely shift register datapath that simply matches the DATA transform latency and are recombined into the Token Stream before output.

B.9.8.5 Transpose RAM

The transpose RAM 444 "tram", is similar in many ways to the inverse zig-zag RAM 441in the way it handles a Token Stream. The width of Tokens handled (22 bits) and the re-ordering performed are different, but otherwise they work in the same way and actually share much of their control logic. Again, rows are additionally re-ordered for the requirements of the following IDCT dimension as well as the fundamental swapping of columns into rows.

B.9.8.6 1-Dimensional Transform - 2nd Dimension

The next block shown is another instance of a single dimension IDCT transform and is identical in every way to the first dimension. At the output of this dimension there interface latch is of the "front" type, meaning that all inputs arrive onto transistor gates to allow safe operation when this block (at the front of the IDCT) is on a separate power supply regime from the one preceding it. This block works by parsing a Token Stream and passing non-DATA Tokens straight through. When a DATA Token is found, a count is started of the number of extensions found after the header. If the extension bit is found to be "0" when the count does not equal 63, an error signal is generated (which goes to the event logic) and depending on the state of the mask bit for that event, "decheck" will either be stopped (i.e., no longer accept input or generate output) or will begin error recovery. The recovery mechanism for "deficient" errors uses the counter to control the insertion of the correct number of extensions into the Token Stream (the value inserted is always "0"). Obviously, input is not accepted whilst this insertion proceeds. When it is found that the extension bit is not "0" on the 64th extension, a "supernumerary" error is generated, the DATA Token is completed by forcing the extension bit to "0", and all succeeding words with the extension bit set to "1" are deleted from the Token Stream by continuing to accept data but invalidating the output.

Note that the two error signals are not persistent (unless the block is stopped) i.e., the error signal only remains active from the point when an error is detected until recovery is complete. This is a minimum of one complete cycle and can persist forever in the case of a infinitely supernumerary DATA Token.

B.9.9.3 "Izz" and "tram" - Reordering RAMs

The "izz" 441 (inverse zig-zag RAM) and the "tram" 444 (transpose RAM) are considered here together since they both perform a variation on the same function and they have more similarities than differences. Both these blocks take a Token Stream and re-order the extensions of a DATA Token whilst passing through all other Tokens unchanged. The widths of the extensions handled and the sequences of the

1,3,5,7,0,2,4,6) rather than (0,1,2,3,4,5,6,7)) because of the requirements of the IDCT transform 1-dimensional blocks.

Transpose address sequence generation is quite straightforward algorithmically. Straight transpose sequence generation simply requires the generation of row and column addresses separately, both implemented with counters. The row re-ordering requirement simply means that row addresses are generated with a simple specific state machine rather than a natural counter.

Inverse zig-zag sequences are rather less straightforward to generate algorithmically. Because of this fact, a small ROM is used to hold the entire 64 6 bit values of address, this being addressed with row and column counters which can be swapped in order to change between horizontal and vertical scan modes. A ROM based generator is very quick to design and it further has the advantage that it is trivial to implement a forward zig-zag (ROM re-program) or to add other alternative sequences in the future.

B.9.9.4 "Oned" - Single Dimension IDCT Transform

This block has a pipeline depth of 20 stages and the pipeline is rigid when stalled. This rigidity greatly simplifies the design and should not unduly affect overall dynamics since the pipeline depth is not that great and both dimensions come after a RAM which provides a certain amount of buffering.

The block follows the standard structure, but has separate paths internally for DATA Token extensions (which are to be processed) and all other items which should be passed through unchanged. Note that the schematic is drawn in a particular way. First, because of the requirements to group together all the datapath logic and second, to allow automatic compiled code generation (this explains the control logic at the top level).

Tokens are parsed as normal and then DATA extensions, and other values, are routed respectively through two different parallel paths before being re-combined with a multiplexer output. Some of latches are of the muxing type, i.e., they can be conditionally loaded from more than one source. All the latches are of the enabled type, i.e., there are separate clock and enable inputs. This means that it is easy to generate enable signals with the correct timing, rather than having to consider issues of skew that would arise if a generated clock scheme was adopted.

The main arithmetic elements required are as follows.

- a number of fixed coefficient multipliers (carry-save output)
- carry-save adders
- carry-save subtractors
- resolving adders
- resolving adder/subtractors

All arithmetic is performed in two's complement representation. This can either be in normal (resolved) form or in carry-save form (i.e., two numbers whose sum represents the actual value). All numbers are resolved before storage and only one resolving operation is performed per pipeline stage since this is the most expensive operation in terms of time. The resolving operations performed here all use simple ripple-carry. This means that the resolvers are quite small, but relatively slow. Since the resolutions dominate the total time in each stage, there is obviously an opportunity to speed up the entire transform by employing fast resolving arithmetic units.

B.9.9.5 "Ras" - Rounding and Saturation

In the present invention, the "ras" block has the task of taking 22-bit fixed point numbers from the output of the second dimension "oned" and turning these into the correctly rounded and saturated 9-bit signed integer results required. This block also performs the necessary divide-by-4 inherent in the scheme (the 2/N term) and to further divide-by-2 required to compensate for the $\sqrt{2}$ pre-scaling performed in each of the two dimensions. This division by 8 implies that the fixed point position is

Addr. (hex)	Bit num.	Register Name
0x0	7..1	not used
	0	TRAM keyhole address
0x1	7..0	
0x2	7..0	TRAM keyhole data
0x3	7..0	TRAM keyhole data[a]
0x4	7..0	IZZ keyhole address
0x5	7..0	IZZ keyhole data
0x6	7..3	not used
	2	ipfsnoop test select
	1	ipfsnoop valid
	0	ipfsnoop accept
0x7	7..6	not used
	5..0	ipfsnoop bits[21:16]
0x8	7..0	ipfsnoop bits[15:8]
0x9	7..0	ipfsnoop bits[7:0]
0xA	7..3	not used
	2	d2snoop test select
	1	d2snoop valid
	0	d2snoop accept
0xB	7..6	not used
	5..0	d2snoop bits[21:16]
0xC	7..0	d2snoop bits[15:8]
0xD	7..0	d2snoop bits[7:0]
0xE	7	outsnoop test select
	6	outsnoop valid
	5	outsnoop accept
	4..2	not used
0xE	1..0	outsnoop data[9:8]
0xF	7..0	outsnoop data[7:0]

Table B.9.1 IDCT Test Address Spac

a. Repeated address

block at the front of the IDCT if incorrect DATA Tokens are detected. The single control bit is "vscan" which is set if it is required to operate the IDCT with the output vertically scanned. This bit, therefore, controls the "izz" block. All the event logic and the memory mapped control bit are located in the block "idctregs".

From the point of view of the IDCT, these registers are located in the following locations. The tristate i/o wires n_derrd and n-serrd are used to read and write to these locations as appropriate.

Addr. (hex)	Bit num.	Register Name
0x0	7..1	not used
	0	vscan

Table B.9.2 IDCT Control Register Address Space

Addr. (hex)	Bit name	Register Name
0x0	n_derrd	idct_too_few_event
	n_serrd	idct_too_many_event
0x1	n_derrd	idct_too_few_mask
	n_serrd	idct_too_many_mask

Tabl B.9.3 IDCT Event Address Spac

by using latches. However, this scheme requires the user to consider several factors.

- control shift-register must now produce control signals of both phases for use as enables (i.e., need to use latches in this shift-register)
- timing analysis complicated by use of latches
- the "t_postc" will no longer automatically produce compiled code since one latch outputs to another latch of the same phase (because of the timing of the enables this is not a problem for the circuit)

Nonetheless, the area saved by the use of latches makes it worthwhile to accept these factors in the present invention.

B.9.11.1.4 Microprocessor interfaces

Due to the nature of this interface, there is a requirement for latches (and resynchronizers) in the Event and register block "idctregs" and in the keyhole logic for RAM cores.

B.9.11.1.5 JTAG Test Control

These standard blocks make use of latches.

B.9.11.2 Circuit Design Issues

Apart from the work done in the design of the library cells that were used in the IDCT design (standard cells, datapath library, RAMs, ROMs, etc.) there is no requirement for any transistor level circuit design in the IDCT. Circuit simulations (using Hspice) were performed of some of the known critical paths in the transform datapath and Hspice was also used to verify the results of the Critical Path Analysis (CPA) tool in the case of paths that were close to the allowed maximum length.

Note that the IDCT is fully static in normal operation (i.e., we can stop the system clocks indefinitely) but there are dynamic nodes in scanable latches which will decay when test clocks are stopped (or very slow). Due to the non-restored nature of some nodes which exhibit a Vt drop (e.g., mux outputs) the IDCT will not be "micro-power" when static.

final layout checks.

The initial work on the transform architecture was done in C, both full-precision and bit-accurate integer models were developed. Various tests were performed on the bit-accurate model in order to prove the conformance to the H.261 accuracy specification and to measure the dynamic ranges of the calculations within the transform architecture.

The design progressed in many cases by writing an M behavioral description of sub-blocks (for example, the control of datapaths and RAMs). Such descriptions were simulated in Lsim before moving onto the design of the schematic description of that block. In some cases (e.g., RAMs, clock generators) the behavioral descriptions were still used for top-level simulations.

The strategy for performing logic simulation was to simulate the schematics for everything that would simulate adequately at that level. The low-level library cells (i.e., zcells and kdplib) were mainly simulated using their behavioral descriptions since this results in far smaller and quicker simulations. Additionally, the behavioral library cells provide timing check features which can highlight some circuit configuration problems. As a confidence check, some simulations were performed using the transistor descriptions of the library cells. All the logic simulations were in the zero-delay manner and, therefore, were intended to verify functional performance. The verification of the real timing behavior is done with other techniques.

Lsim switch-level simulations (with RC_Timing mode being used) were done as a partial verification of timing performance, but also provide checks for some other potential transistor level problems (e.g., glitch sensitive circuits).

The main verification technique for checking timing problems was the use of the CPA tool, the "path" option for "datechk". This was used to identify the longer signal "decheck" block to the "ip_fmt" block and the latter from the first "oned" block to the "ras" block.

Access to snoopers is possible by accessing the appropriate memory mapped locations in the normal manner. The same is true of the RAM cores (using the "ramtest" input as appropriate). The scan paths are accessed through the JTAG port in the normal way.

Each of the blocks is now discussed with reference to the various test issues.

B.9.13.1 "Decheck"

This block has the standard structure (see Figure 139) where two latches for the input and output two-wire interfaces surround a processing block. As usual, no scan is provided to the two-wire latches since these simply pass on data whenever enabled and have no depth of logic to be tested. In this block, the "control" section consists of a 1-stage pipeline of zcells which are all on scanpath "a". The logic in the control section is relatively simple, the most complex path is probably in the generation of the DATA extension count where a 6-bit incrementer is used.

B.9.13.2 "Izz"

This block is a variant of the standard structure and includes a RAM core block added to the two-wire interface latches and the control section. The control section is implemented with zcells and a small ROM used for address sequence generation. All the zcells are on scanpath "a" and there is access to the ROM address and data via zcell latches. There is also further logic, e.g., for the generation of numbers plus the ability to increment or decrement. In addition, there is a 7-bit full adder used for read address generation. The RAM core is accessible through keyhole registers, via the microprocessor interface, see Table B.9.1.

B.9.13.3 "lp_fmt"

This block again has the standard structure. Control logic is implemented with some rather simple zcell logic (all on scanpath "a") but the latching and shifting/muxing

SECTION B.10 Introduction

B.10.1 Overview of the Temp ral D c der

The internal structure of the Temporal Decoder, in accordance with the invention, is shown in Figure 142.

All data flow between the blocks of the chip (and much of the data flow within blocks) is controlled by means of the usual two-wire interfaces and each of the arrows in Figure 142 represents a two-wire interface. The incoming token stream passes through the input interface 450 which synchronizes the data from the external system clock to the internal clock derived from the phase-locked-loop (ph0/ph1). The token stream is then split into two paths via a Top Fork 451; one stream passes to the Address Generator 452 and the other to a 256 word FIFO 453. The FIFO buffers data while data from previous I or P frames is fetched from the DRAM and processed in the Prediction Filters 454 before being added to the incoming error data from the Spatial Decoder in the Prediction Adder 455 (P and B frames). During MPEG decoding, frame reordering data must also be fetched for I and P frames so that the output frames are in the correct order, the reordered data being inserted into the stream in the Read Rudder block 456.

The Address Generator 452 generates separate addresses for forward and backward predictions, reorder, read and write-back, the data which is written back being split from the stream in the Write Rudder block 457. Finally, data is resynchronized to the external clock in the Output Interface Block 458.

All the major blocks in the Temporal Decoder are connected to the internal microprocessor interface (UPI) bus. This is derived from the external microprocessor interface (MPI) bus in the Microprocessor Interface block 459. This block has address decodes for the various blocks in the chip associated with it. Also associated with the microprocessor interface is the event logic.

The rest of the logic of the Temporal Decoder is

SECTION B.11 Clocking, Test and Related Issues

B.11.1 Clock Regimes

Before considering the individual functional blocks within the chip, it is helpful to have an appreciation of the clock regimes within the chip and the relationship between them.

During normal operation, most blocks of the chip run synchronously to the signal pllsysclk from the phase-locked-loop (PLL) block. The exception to this is the DRAM interface whose timing is governed by the need to be synchronous to the iftime sub-block, which generates the DRAM control signals (notwe, notoe, notcas, notras). The core of this block is clocked by the two-phase non-overlapping clocks clk0 and clk1, which are derived from the quadrature two-phase clocks supplied independently from the PLL cki0, cki1 and clkq0, ckq1.

Because the clk0, clk1 DRAM interface clocks are asynchronous to the clocks in the rest of the chip, measures have been taken to eliminate the possibility of metastable behavior (as far as practically possible) at the interfaces between the DRAM interface and the rest of the chip. The synchronization occurs in two areas: in the output interfaces of the Address Generator (addrgen/predread/psgsync, addrgen/ip_wrt2/sync18 and addrgen/ip_rd2/sync18) and in the blocks which control the "swinging" of the swing-buffer RAMs in the DRAM Interface (see section on the DRAM Interface). In each case, the synchronization process is achieved by means of three metastable-hard flip-flops in series. It should be noted that this means that clk0/clk1 are used in the output stages of the Address Generator.

In addition to these completely asynchronous clock regimes, there are a number of separate clock generators which generate two-phase non-overlapping clocks (ph0, ph1) from pllsysclk. The Address Generator, Prediction Filters and DRAM Interface each have their own clock generators; the remainder of the chip is run off a common clock

pllselect	override	Mode
0	0	pllsysclk is connected directly to external sysclk, bypassing the PLL; DRAM Interface clocks (cki0, cki1, ckq0, ckq1) are controlled directly from the pins ti and tq.
0	1	Override mode - ph0 and ph1 clocks are controlled directly from pins tphoish and tph1ish; DRAM Interface clocks (cki0, cki1, ckq0, ckq1) are controlled directly from the pins ti and tq.
1	0	Normal operation. pllsysclk is the clock generated by the PLL; DRAM Interface clocks are generated by the PLL.
1	1	External resistors connected to ti and tq are used instead of the internal resistors (debug only).

Table B.11.1 Clock Control Modes

B.11.3 The Two-wire Interface

The overall functionality of the two-wire interface is described in detail in the Technical Reference. However, the two-wire interface is used for all block-to-block communication within the Temporal Decoder and most blocks consist of a number of pipeline stages, all of which are themselves two-wire interface stages. It is, therefore, essential to understand the internal implementation of the two-wire interface in order to be able to interpret many of the schematics. In general, these internal pipeline stages are structured as shown in Figure 143.

Location	Type
addrgen/vec_pipe/snoopz31	Snooper
addrgen/cnt_pipe/midsnp	Snooper
addrgen/cnt_pipe/endsnp	Snooper
addrgen/precread/snoopz44	Snooper
addrgen/ip_wrt2/superz10	Super Snooper
addrgen/ip_rd2/superz10	Super Snooper

Table B.11.2 Snoopers in Temporal Decoder.

Location	Type
dramx/dramif/ifsnoops/snoopz15 (fsnp)	Snooper
dramx/dramif/ifsnoops/snoopz15 (bsnp)	Snooper
dramx/dramif/ifsnoops/superz9	Super Snooper
wrudder/superz9	Super Snooper
pflts/fwdflt/dimbuff/snoopk13	Snooper
pflts/bwdflt.dimbuff/snoopk13	Snooper
pflts/snoopz9	Snooper

Table B.11.2 Snoopers in Temporal Decoder

Details on the use of both Snoopers are contained in the test section. Details of the operation of the JTAG interface are contained in the JTAG document.

the token stream from the token input port (via topfork), and its output to the DRAM interface consists of addresses and other information, controlled by a request/acknowledge protocol.

The principal sections of the address generator are:

- token decode
- block counting and generation of the DRAM block address
- conversion of motion vector data into an address offset
- request and address generator for prediction transfers
- reorder read address generator
- write address generator

B.12.2.1 Token Decode (tokdec)

In the Token Decoder, tokens associated with coding standards, frame and block information and motion vectors are decoded. The information extracted from the stream is stored in a set of registers which may also be accessed via the upi. The detection of a DATA token header is signalled to subsequent blocks to enable block counting and address generation. Nothing happens when running JPEG.

List of tokens decoded:

- CODING_STANDARD
- DATA
- DEFINE_MAX_SAMPLING
- DEFINE_SAMPLING
- HORIZONTAL_MBS
- MVD_BACKWARDS
- MVD_FORWARDS
- PICTURE_START
- PICTURE_TYPE
- PREDICTION_MODE

This block also combines information from the request generators to control the toggling of the frame pointers and to stall the input stream. The stream is stalled when a new frame appears at the input (in the form of a group of blocks. This is eleven macroblocks wide and three deep, and a picture is always two groups wide. The token decoder xtracts the CIF bit from the PICTURE_TYPE token and passes this to the macroblock counter to instruct it to count groups of blocks. Instances of too few or too many blocks per component will provoke similar reactions as above.

B.12.2.3 Block Calculation (blkcalc)

The Block calculation converts the macroblock and block-within-macroblock coordinates into coordinates for the block's position in the picture, i.e., it knocks out the level of hierarchy. This, of course, has to take into account the sampling ratios of the different color components.

B.12.2.4 Base block Address (bsblkadr)

The information from the **blkcalc**, together with the color component offsets, is used to calculate the block address within the linear DRAM address space. Essentially, for a given color component, the linear block address is the number of blocks down times the width of the picture plus the number of blocks long. This is added to the color component offset to form the base block address.

B.12.2.5 Vector Offset (vec_pipe)

The motion vector information presented by the token decoder is in the form of horizontal and vertical pixel offset coordinates. That is, for each of the forward and backward vectors there is an (x,y) which gives the displacement in half-pixels from the block being formed to the block from which it is being predicted. Note that these coordinates may be positive or negative. They are first scaled according to the sampling of each color component, and used to form the block and new pixel offset coordinates.

In Figure 145, the shaded area represents the block that is being formed. The dotted outline is the block from which it is being predicted. The big arrow shows the block offset - the horizontal and vertical vector to the DRAM

B.12.2.8 Offsets

The DRAM is configured as two frame stores, each of which contains up to three color components. The frame store pointers and the color component offsets within each frame must be programmed via the upi.

B.12.2.9 Snoopers

In the present invention, snoopers are positioned as follows:

·Between **blkcalc** and **bsblkadr** - this interface comprises the horizontal and vertical block coordinates, the appropriate color component offset and the width of the picture in blocks (for that component).

·After **bsblkadr** - the base block address.

·After vec_pipe - the linear block offset, the pixel offset within the block, together with information on the prediction mode, color component and H.261 operation.

·After **Inblkad3** - the physical block address, as described under "Prediction Requests".

Super snoopers are located in the reorder read and write request generators for use during testing of the external DRAM. See the DRAM Interface section for all the details.

B.12.2.10 Scan

The **addrgen** block has its own scan chain, the clocking of which is controlled by the block's own clock generator (**adclkgen**). Note that the request generators at the back end of the block fall within the DRAM interface clock regime.

B.12.3 **Prediction Filters

The overall structure of the Prediction Filters, in accordance with the present invention, is shown in Figure 146. The forward and backward filters are identical and filter the MPEG forward and backward prediction blocks. Only the forward filter is used in H.261 mode (the h261_on input of the backward filter should be permanently low because H.261 streams do not contain backward predictions). The entire Prediction Filters block is composed of horizontal groups of three (see notes on the Dimension Buffer, below) there is no need to treat the first and last pixels in a row differently. The control and the counting of the pixels within a row is performed by the control logic associated with each 1-D filter. It should be noted that the result has not been divided by 4. Division by 16 (shift right by 4) is performed at the input of the Prediction Filters Adder (Section B.12.4.2) after both horizontal and vertical filtering has been performed, so that arithmetic accuracy is not lost. Registers DA, DD and DF pass control information down the pipeline. This includes h261_on and last_byte.

Of the other blocks found in the Prediction Filter, the function of the Formatter is merely to ensure that data is presented to the x-filter in the correct order. It can be seen above that this merely requires a three-stage shift register, the first stage being connected to the input of register C, the second to register B and the third to register A.

Between the x and y filters, the Dimension Buffer buffers data so that groups of three vertical pixels are presented to the y-filter. These groups of three are still processed horizontally, however, so that no transposition occurs within the Prediction Filters. Referring to Figure 149, the sequence in which pixels are output from the Dimension Buffer is illustrated in Table B.12.1.

B.12.3.1.2 MPEG Operation

During MPEG operation, a Prediction Filter performs a simple half pel interpolation:

$$F_i = \frac{x_i + x_{i+1}}{2}(0 \le i \le 8,\text{half}\,pel)$$

$$F_i = x_i(0 \le i \le 7,\text{integer}\,pel)$$

This is the default filter operation unless the h261_on input is low. If the signal dim into a 1-D filter is low then integer pel interpolation will be performed. Accordingly, if h261_on is low and xdim and ydim are low, all pixels are passed straight through without filtering. It is an obvious requirement that when the dim signal into a 1-D filter is high, the rows (or columns) will be 8 pixels wide (or high). This is summarized in Table B.12.2. Referring to Figure 148, "1-D Prediction Filter,", the

h261_on	xdim	ydim	Function
0	0	0	$F_i = x_i$
0	0	1	MPEG 8x9 block
0	1	0	MPEG 9x8 block
0	1	1	MPEG 9x9 block
1	0	0	H.261 Low-pass Filter
1	0	1	Illegal
1	1	0	Illegal
1	1	1	Illegal

Table B.12.2 1-D Filter Operation

(rounded towards positive infinity).

The prediction mode can only change between blocks, i.e., at power-up or after the fwd_1st_byte and/or bwd_1st_byte signals are active, indicating the last byte of the current prediction block. If the current block is a forward prediction then only fwd_1st_byte is examined. If it is a backward prediction then only bwd_1st_byte is examined. If it is a bidirectional prediction, then both fwd_1st_byte and bwd_1st_byte are examined.

The signals fwd_on and bwd_on determine which prediction values are used. At any time, either both or neither of these signals may be active. At start-up, or if there is a gap when no valid data is present at the inputs of the block, the block enters a state when neither signal is active.

Two criteria are used to determine the prediction mode for the next block: the signals fwd_ima_twin and bwd_ima_twin, which indicate whether a forward or backward block is part of a bidirectional prediction pair, and the buses fwd_p_num[1:0] and bwd_p_num[1:0]. These buses contain numbers which increment by one for each new prediction block or pair of prediction blocks. These blocks are necessary because, for example, if there are two forward prediction blocks followed by a bidirectional prediction block, the DRAM interface can fetch the backward block of the bidirectional prediction sufficiently far ahead so that it reaches the input of the Prediction Filters Adder before the second of the forward prediction blocks. Similarly, other sequences of backward and forward predictions can get out of sequence at the input of the Prediction Filters Adder. Thus, the next prediction mode is determined as follows:

1)If valid forward data is present and fwd_ima_twin is high, then the block stalls until valid backward data arrives with bwd_ima_twin set and then it goes through the blocks averaging each pair of prediction values.

B.12.4 Prediction Adder and FIFO

The prediction adder (padder) forms the predicted frame by adding the data from the prediction filters to the error data. To compensate for the delay from the input through the address generator, DRAM interface and prediction filters, the error data passes through a 256 word FIFO (sfifo) before reaching padder.

The CODING_STANDARD, PREDICTION_MODE and DATA Tokens are decoded to determine when a predicted block is being formed. The 8-bit prediction data is added to the 9-bit two's complement error data in the DATA Token. The result is restricted to the range 0 to 255 and passes to the next block. Note that this data restriction also applies to all intra-coded data, including JPEG.

The prediction adder of the present invention also includes a mechanism to detect mismatches in the data arriving from the FIFO and the prediction filters. In theory, the amount of data from the filters should exactly correspond to the number of DATA Tokens from the FIFO which involve prediction. In the event of a serious malfunction, however, padder will attempt to recover.

The end of the data blocks from the FIFO and filters are marked, respectively, by the in_extn and fl_last inputs. Where the end of the filter data is detected before the end of the DATA Token, the remainder of the token continues to the output unchanged. If, on the other hand, the filter block is longer than the DATA Token, the input is stalled until all the extra filter data has been accepted and discarded.

There is no snooper in either the FIFO or the prediction adder, as the chip can be configured to pass data from the token input port directly to these blocks, and to pass their output directly to the token output port.

B.12.5 Write and Read Rudders

B.12.5.1 The Write Rudder (wrudder)

The Write Rudder passes all tokens coming from the Prediction Adder on to the Read Rudder. It also passes all

B.12.5.2 The Read Rudder (rrudder)

The Read Rudder of the present invention has three functions, the two major ones relating to picture sequence reordering in MPEG:

1) To insert data which has been read-back from the external frame store into the token stream at the correct places.

2) To reorder picture header information in I and P pictures.

3) To detect the end of a token stream by detecting the FLUSH token (see Section B.12.1, "Top Fork").

The structure of the Read Rudder is illustrated in Figure 150. The entire block is made from standard two-wire interface technology. Tokens in the input interface latches are decoded and these decodes determine the operation of the block:

Token Name	Function in Read Rudder
FLUSH	Signals to Top Fork.
CODING_STANDARD	Reordering is inhibited if the coding standard is not MPEG.
SEQUENCE_START	The read-back data for the first picture of a reordered sequence is invalid.
PICTURE_START	Signals that the current output FIFO must be swapped (I or P pictures). The first of the picture header tokens.
PICTURE_END	All tokens above the picture layer are allowed through
TEMPORTAL_REFERENCE	The second of the picture header tokens.
PICTURE_TYPE	The third of the picture header tokens.
DATA	When reordering, the contents of DATA tokens are replaced with reordered data.

Table B.12.4 Tokens decoded by th R ad Rudd r

is encountered, after which FIFO 2 becomes the current output FIFO. Referring back to Section B.12.5.2.1, it can be seen that at this stage the three picture header tokens, PICTURE_START, TEMPORAL_REFERENCE and PICTURE_START are retained in FIFO 1. The current output FIFO is swapped every time a picture start code is encountered in an I or P frame. Accordingly, the three picture header tokens are stored until the next I or P frame, at which time they will become associated with the correctly reordered data. B pictures are not reordered and, hence, pass through without any tokens being discarded. All tokens in the first picture, including PICTURE_END are discarded.

During I and P pictures, the data contained in DATA Tokens in the token stream is replaced by reordered data from the DRAM Interface. During the first picture, "reordered" data is still present at the reordered data input because the Address Generator still requests the DRAM Interface to fetch it. This is considered garbage and is discarded.

from the DRAM, one of each of Luminance (Y) and the Red and Blue color difference data (Cr and Cb respectively).

The operation of the generic features of the DRAM Interface is described in the Spatial Decoder document. The following section describes the features peculiar to the Temporal Decoder.

B.13.2 The Temporal Decoder DRAM Interface

As mentioned in section B.13.1, the Temporal Decoder has four swing buffers: two are used to read and write decoded Intra and Predicted (I and P) picture data and these operate as described above. The other two are used to fetch prediction data.

In general, prediction data will be offset from the position of the block being processed as specified by motion vectors in x and y. Thus, the block of data to be fetched will not generally correspond to the block boundaries of the data as it was encoded (and written into the DRAM). This is illustrated in Figures 151 and 25, where the shaded area represents the block that is being formed. The dotted outline shows the block from which it is being predicted. The address generator converts th address specified by the motion vectors to a block offset (a whole number of blocks), as shown by the big arrow, and a pixel offset, as shown by the little arrow.

In the address generator, the frame pointer, base block address and vector offset are added to form the address of the block to be fetched from the DRAM. If the pixel offset is zero, only one request is generated. If there is an offset in either the x or y dimension, then two requests are generated - the original block address and the one either immediately to the right or immediately below. With an offset in both x and y, four requests are generated. For each block which is to be fetched, the address generator calculates start and stop addresses parameters and passes these to the DRAM interface. The use of these start and stop addresses is best illustrated by an example, as outlined below.

the x inverse stop value forming the 3 LSBs. In this case, while the DRAM Interface is reading address 9 in the external DRAM, the swing buffer address is zero. The swing buffer address register is then incremented as the external DRAM address register is incremented, as illustrated in Table B.13.1:

Table B.13.1 Illustration of Prediction Addressing

Ext DRAM Address	Swing Buff Address	Ext DRAM Ad. (Binary)	Swing Buff Ad. (Binary)
9 = y-start, x-start	0 = $\overline{\text{y-stop}}$, $\overline{\text{x-stop}}$	001 001	000 000
10	1	111 110	000 001
11	2	001 011	000 010
15	6	001 111	000 110
17 = y+1, x-start	8 = y+1, $\overline{\text{x-stop}}$	010 001	001 000
18	9	010 010	001 001

The discussion thus far has centered on an 8 bit DRAM Interface. In the case of a 16 or 32 bit interface, a few minor modifications must be made. First, the pixel offset vector must be "clipped" so that it points to a 16 or 32 bit boundary. In the example we have been using, for block A, the first DRAM read will point to address 0, and data in addresses 0 through 3 will be read. Next, the unwanted data must be discarded. This is performed by writing all the data into the swing buffer (which must now be physically bigger than was necessary in the 8 bit case) and reading with an offset. When performing MPEG half-pel interpolation, 9 bytes in x and/or y must be read from the DRAM Interface. In this case, the address generator provides the appropriate start and stop addresses and some additional logic in the DRAM Interface is used, but there

SECTION B.14 UPI Documentation

B.14.1 Introducti n

This document is intended to give the reader an appreciation of the operation of the microprocessor interface in accordance with the present invention. The interface is basically the same on both the SPATIAL DECODER and the Temporal Decoder, the only difference being the number of address lines.

The logic described here is purely the microprocessor internal logic. The relevant schematics are:

UPI
UPI101
UPI102
DINLOGIC
DINCELL
UPIN
TDET
NONOVRLP
WRTGEN
READGEN
VREFCKT

The circuits UPI, UPI101, UPI102 are all the same except that the UPI01 has a 7 bit address input with the 8th bit hardwired to ground, while the other two have an 8 bit address input.

Input/Output Signals

The signals described here are a list of all the inputs and outputs (defined with respect to the UPI) to the UPI module with a note detailing the source or destination of these signals:

NOTRSTInputGlobal chip reset, active low, from Pad Input Driver

E1InputEnable signal 1, active low, from the Pad Input Driver (Schmitt).

E2InputEnable signal 2, active low, from the Pad Input Driver (Schmitt).

RNOTWInputRead not Write signal from the Pad Input

The other implication of not having a clock in the UPI is that some internal timing is self timed. That is, the delay of some signals is controlled internally to the UPI block.

The overall function of the UPI is to take the address, data and enable and read/write signals from the outside world and format them so that they can drive the internal circuits correctly. The internal signals that define access to the memory map are INT_RNOTW_INT_ADDR[...], INTDBUS[...] and READ_STR and WRITE_STR. The timing relationship of these signals is shown below for a read cycle and a write cycle. It should be noted that although the datasheet definition and the following diagram always shows a chip enable cycle, the circuit operation is such that the enable can be held low and the address can be cycled to do successive read or write operations. This function is possible because of the address transition circuits.

Also, the presence of the INT_RNOTW and the READ_STR, WRITE_STR does reflect some redundancy. It allows internal circuits to use either a separate READ_STR and WRITE_STR (and ignore INT_RNOTW) or to use the INT_RNOTW and a separate Strobe signal (Strobe signal being derived from OR of READ_STR and WRITE_STR).

The internal databus is precharged High during a read cycle and it also has resistive pullups so that for extended periods when the internal data bus is not driven it will default to the 0XFF condition. As the internal databus is the inverse of the data on the pins, this translates to 0x00 on the external pins, when they are enabled. This means that, if any external cycle accesses a register or a bit of a register which is a hole in the memory map, then the output data id determinate and is Low.

Circuit Details:

UPIN -

This circuit is the overall change detect block. It contains a sub-circuit called TDET which is a single bit the input pad is latched when the signal DATAHOLD is high and when both Enable1 and Enable2 are low. The tristate driver drives the internal data bus whenever the internal signal INT_RNOTW is low. The internal databus precharge transistor and the bus pullup are also included in this module.

WRTGEN-

This module generates the WRITE_STR, and the latch signal DATAHOLD for the data latches. The write strobe is a self timed signal, however, the self time delay is defined in the VREFCKT. The output from the timing circuit RESETWRITE is used to terminate the WRITE_STR signal. It should be noted that the actual write pulse which writes a register only occurs after an access cycle is concluded. This is because the data input to the chip is sampled only on the back edge of the cycle. Hence, data is only valid after a normal access cycle has concluded.

READGEN-

This circuit, as its name suggests, generates the READ_STR and it also generates the PRECH signal which is used to precharge the internal databus. The PRECH signal is also a self timed signal whose period is dependant on VREFCKT and also on the voltage on the internal databus. The READ_STR is not self timed, but lasts from the end of the precharge period until the end of the cycle. The precharge circuitry uses inverters with their transfer characteristic biased so that they need a voltage of approximately 75% of supply before they invert. This circuit guarantees that the internal bus is correctly precharged before a READ_STR begins. In order to stop a PRECH pulse tending to zero width if the internal bus is already precharged, the timing circuit guarantees a minimum, width via the signal RESETREAD.

VREFCKT-

The VREFCKT is the only circuit which controls the self timing of the interface. Both the delays, 1/Width of WRITE_STR and 2/Width of PRECH, are controlled by a current

SECTION C.1 Overview

C.1.1. Introduction

The structure of the image Formatter, in accordance with the present invention, is shown in Figure 155. There are two address generators, one for writing and one for reading, a buffer manager which supervises the two address generators and which provides frame-rate conversion, a data processing pipeline, including both vertical and horizontal unsamplers, color-space conversion and gamma correction, and a final control block which regulates the output of the processing pipeline.

C.1.2 Buffer manager

Tokens arriving at the input to the Image Formatter are buffered in the FIFO and then transferred into the buffer manager. This block detects the arrival of new pictures and determines the availability of a buffer in which to store each picture. If there is a buffer available, it is allocated to the arriving picture and its index is transferred to the write address generator. If there is no buffer available, the incoming picture will be stalled until one becomes available. All tokens are passed on to the write address generator.

Each time the read address generator receives a VSYNC signal from the display system, a request is made to the buffer manager for a new display buffer index. If there is a buffer containing complete picture data, and that picture is deemed ready for display, then that buffer's index will be passed to the display address generator. If not, the buffer manager sends the index of the last buffer to be displayed. At start-up, zero is passed as the index until the first buffer is full.

A picture is ready for display if its number (calculated as each picture is input) is greater than or equal to the picture number which is expected at the display (presentation number) given the encoding frame rate. The expected number is determined by counting picture clock of the three components that are compatible with the page structure of the DRAM. The addresses provided to the DRAM interface are page and line addresses along with block start and block end counts.

C.1.5 Output Pipeline

Data from the DRAM interface feeds the output pipeline. The three component streams are first vertically interpolated, then horizontally interpolated. Following the interpolators, the three components should be of equal ratios (4:4:4), and are passed through the color-space converter and color lookup tables/gamma correction. The output interface may hold the streams at this point until the display has reached an HSYSC. Thereafter, output controller directs the three components into one, two or three 8-bit buses, multiplexing as necessary.

C.1.6 Timing Regimes

There are basically two principal timing regimes associated with the Image Formatter. First, there is a system clock, which provides timing for the front end of the chip (address generators and buffer manager, plus the front end of the DRAM interface). Second, there is a pixel clock which drives all the timing for the back end (DRAM interface output, and the whole of the output pipeline).

Each of the two aforementioned clocks drives a number of on-chip clock generators. The FIFO, buffer manager and read address generator operate from the same clock (DΦ) with the write address generator using a similar, but separate clock (WΦ). Data is clocked into the DRAM interface on an internal DRAM interface clock, (outΦ). DΦ, WΦ and outΦ are all generated from syscik.

Read and write addresses are clocked in the DRAM interface by the DRAM interface's own clock.

Data is read out of the DRAM interface on bifRΦ, and is transferred to the section of the output pipeline named "bushy_ne" (north-east - by virtue of its physical location) which operates on clocks denoted by NEΦ. The section of the pipeline from the gamma RAMs onward is

SECTION C.2 Buffer Management

C.2.1. Introducti n

The purpose of the buffer management block, in accordance with the present invention, is to supply the address generators with indices identifying any of either two or three external buffers for writing and reading of picture data. The allocation of these indices is influenced by three principal factors, each representing the effect of one of the timing regimes in operation. These are the rate at which picture data arrives at the input to Image Formatter (coded data rate), the rate at which data is displayed (display data rate), and the frame rate of the encoded video sequence (presentation rate).

C.2.2 Functional Overview

A three-buffer system allows the presentation rate and the display rate to differ (e.g., 2-3 pulldown), so that frames are either repeated or skipped as necessary to achieve the best possible sequence of frames given the timing constraints of the system. Pictures which present some difficulty in decoding may also be accommodated in a similar way, so that if a picture takes longer than the available display time to decode, the previous frame will be repeated while everything else "catches up". In a two-buffer system, the three timing regimes must be locked - it is the third buffer which provides the flexibility for taking up the slack.

The buffer manager operates by maintaining certain status information associated with each external buffer. This includes flags indicating if the buffer is in use, if it is full of data, or ready for display, and the picture number within the sequence of the picture currently stored in the buffer. The presentation number is also recorded, this being a number which increments every time a picture clock pulse is received, and represents the picture number which is currently expected for display based on the frame rate of the encoded sequence.

C.2.3 Architecture

C.2.3.1 Interfaces

C.2.3.1.1. Interface to bm front

All data is input to the buffer manager from the input FIFO, bm_front. This transfer takes place via a two-wire interface, the data being 8 bits wide plus an extension bit. All data arriving at the buffer manager is guaranteed to be a complete token. This is a necessity for the continued processing of presentation numbers and display buffer requests in the event of significant gaps in the data upstream.

C.2.3.1.2 Interface to waddrgen

Tokens (8 bit data, 1 bit extension) are transferred to the write address generator via a two-wire interface. The arrival buffer index is also transferred on the same interface, so that the correct index is available for address generation at the same time as the PICTURE_START token arrives at waddrgen.

C.2.3.1.3 Interface to dispaddr

The interface to the read address generator comprises two separate two-wire interfaces which can be considered to act as "request" and "acknowledge" signals, respectively. Single wires are not adequate, however, because of the two two-wire-based state machines at either end.

The sequence of events normally associated with the dispaddr interface is as follows. First, dis-paddr invokes a request in response to a vsync from the display device by asserting the drq_valid input to the buffer manager. Next, when the buffer manager reaches an appropriate point in its state machine, it will accept the request and go about allocating a buffer to be displayed. Thereafter, the disp_valid wire is asserted, the buffer index is transferred, and this is typically accepted immediately by dispaddr. Furthermore, there is an additional wire associated with this last two-wire interface (rst_fld) which indicates that the field number associated with the current index must be reset regardless of the previous

The input data register is shown, together with some token decoding hardware attached thereto. The signal vheader at the input to bm_tokdec is used to ensure that the token decoder outputs can only be asserted at a point where a header would be valid (i.e., not in the middle of a token. The rtimd block acts as the output data registers, adjacent to the duplicate input data registers for the next block in the pipeline. This accounts for timing differences due to different clock generators. Signals go and ngo are based on the AND of data valid, accept and not stopped, and are used elsewhere in the state machine to indicate if things are "bunged up" at either the input or the output.

The display index part of this module comprises the two-wire interfaces together with equivalent "go" signals as for data. The rst_fld bit also happens here, this being a signal which, if set, remains high until disp_valid has been high for one cycle. Thereafter, it is reset. In addition, rst_fld is reset after a FLUSH token has caused all of the external buffers to be flagged either as EMPTY or IN_USE by the display buffer. This is the same point at which both picture numbers and presentation number are reset.

There is a small amount of additional circuitry associated with the input data register which appears at the next level up the hierarchy. This circuitry produces a signal which indicates that the input data register contains a value equal to that written into BU_BM_TARGIX and it is used for event generation.

C.2.3.2.2 Index block (bm index)

The Index block consists mainly of the 2-bit registers denoting the various strategic buffer indices. These are arr_buf, the buffer to which arriving picture data is being written, disp_buf, the buffer from which picture data is being read for display, and rdy_buf, the index of the buffer containing the most up to date picture which could be displayed if a buffer was requested by dispaddr. There is also a register containing buf_ix, which is used as a or the previous value incremented by one (or one plus delta, the difference between actual and expected temporal reference, in the case of H.261). This value is supplied by the 8-bit adder present in the block. The first input to this adder is this_pnum, the picture number of the data currently being written.

Buffer Status	Value
EMPTY	00
FULL	01
READY	10
IN_USE	11

Table C.2.1 Buffer Status Values

This needs to be stored separately (in its own master-slave arrangement) so that any of the three buffer picture number registers can be easily updated based on the current (or previous) picture number rather than on their own previous picture number (which is almost always out of date). This_pnum is reset to -1 so that when the first picture arrives it is added to the output from the adder and, hence, the input to the first buffer picture number register, is zero.

Note that in the current version, delta is connected to zero because of the absence of the temporal reference block which should supply the value.

<u>C.2.3.2.4 Presentation Number</u>

The 8-bit presentation number register has an associated presentation flag which is used in the state machine to indicate that the presentation number has changed since it

In the invention, tr is reset to zero and loaded, when appropriate, from the input data register. Similarly, exptr is reset to -1, and is incremented by either 1 or delta during the sequence of temporal reference states. In addition, delta is reset to zero and is loaded with the difference between the other two registers. All three registers are reset after a FLUSH token. The adder in this block is used for calculation of both delta and exptr, i.e., a subtract and an add operation, respectively, and is controlled by the signal delta_calc.

Signal Name	Logic Expression
A	ST_PRES1.presflg.(bstate==FULL).rdytst.(rdy==0).(ix==max)
B	ST_PRES1.presflg.(bstate==FULL).rdytst.(rdy==0).(ix!=max)
C	ST_PRES1.presflg.(bstate==FULL).rdytst.(rdy!=0)
D	ST_PRES1.presflg.!((bstate==FULL).rdytst).(ix==max)
E	ST_PRES1.presflg.!((bstate==FULL).rdytst).(ix!=max)
F	ST_PRES1.presflg
G	ST_DRQ.drq_valid.disp_acc.(rdy==0).(disp!=0)
PP	ST_DRQ.drq_valid.disp_acc.(rdy==0).(disp!=0).fromps
QQ	ST_DRQ.drq_valid.disp_acc.(rdy==0).(disp!=0).fromfl
RR	ST_DRQ.drq_valid.disp_acc.(rdy==0).(disp!=0).!(fromps+fromfl)
H	ST_DRQ.drq_valid.disp_acc.(rdy!=0).(disp!=0)
I	ST_DRQ.drq_valid.disp_acc.(rdy!=0).(disp==0)
J	ST_DRQ.drq_valid.disp_acc.(rdy==0).(disp==0).fromps
NN	ST_DRQ.drq_valid.disp_acc.(rdy==0).(disp==0).fromfl
OO	ST_DRQ.drq_valid.disp_acc.(rdy==0).(disp==0).!(fromps+fromfl)
K	ST_DRQ.!(drq_valid.disp_acc).fromps
LL	ST_DRQ.!(drq_valid.disp_acc).fromfl
MM	ST_DRQ.!(drq_valid.disp_acc).!(fromps+fromfl)
L	ST_TOKEN.ivr.oar.(idr==TEMPORAL_REFERENCE)
SS	ST_TOKEN.ivr.oar.(idr==TEMPORAL_REFERENCE).H261
TT	ST_TOKEN.ivr.oar.(idr==TEMPORAL_REFERENCE).!H261
M	ST_TOKEN.ivr.oar.(idr==FLUSH)
N	ST_TOKEN.ivr.oar.(idr==PICTURE_START)
O	ST_TOKEN.ivr.oar.(idr==PICTURE_END)
P	ST_TOKEN.ivr.oar.(idr==<OTHER_TOKEN>)
JJ	ST_TOKEN.ivr.oar.(idr==<OTHER_TOKEN>).in_extn
KK	ST_TOKEN.ivr.oar.(idr==<OTHER_TOKEN>).!in_extn
Q	ST_TOKEN.!(ivr.oar)

Tabl C.2.2 Signal Names Us d in th State Machine

Register Name	Access	Bits	Reset State	Function
BU_BM_ACCESS	0x10	[0]	1	Access bit for buffer manager
BU_BM_CTL0	0x11	[0]	1	Max buf lsb. 1->3 buffers,0->2
		[1]	1	External picture clock select
BU_BM_TARGET_IX	0x12	[3:0]	0x0	For detecting arrival of picture
BU_BM_PRES_NUM	0x13	[7:0]	0x00	Presentation number
BU_BM_THIS_PNUM	0x14	[7:0]	0xFF	Current picture number
BU_BM_PIC_NUM0	0x15	[7:0]	none	Picture number in buffer 1
BU_BM_PIC_NUM1	0x16	[7:0]	none	Picture number in buffer 2
BU_BM_PIC_NUM2	0x17	[7:0]	none	Picture number in buffer 3
BU_BM_TEMP_REF	0x18	[4:0]	0x00	Temporal reference from stream

Table C.2.3 User-Accessible Registers

Register Name	Address	Bits	Reset State	Function
BU_BM_PRES_FLAG	0x80	[0]	0	Presentation flag
BU_BM_EXP_TR	0x81	[4:0]	0xFF	Expected temporal reference
BU_BM_TR_DELTA	0x82	[4:0]	0x00	Delta
BU_BM_ARR_IX	0x83	[1:0]	0x0	Arrival buffer index
BU_BM_DSP_IX	0x84	[1:0]	0x0	Display buffer index
BU_BM_RDY_IX	0x85	[1:0]	0x0	Ready buffer index
BU_BM_BSTATE3	0x86	[1:0]	0x0	Buffer 3 status
BU_BM_BSTATE2	0x87	[1:0]	0x0	Buffer 2 status
BU_BM_BSTATE1	0x88	[1:0]	0x0	Buffer 1 status
BU_BM_INDEX	0x89	[1:0]	0x0	Current buffer index
BU_BM_STATE	0x8A	[4:0]	0x00	Buffer manager state
BU_BM_FROMPS	0x8B	[0]	0x0	From PICTURE_START flag
BU_BM_FROMFL	0x8C	[0]	0x0	From FLUSH_TOKEN flag

Tabl C.2.4 Test R gisters

C.2.4.1 The Reset State

The reset state is PRES0, with flags set to zero such that the main loop circulated initially.

C.2.4.2 The Main Loop

The main loop of the state machine comprises the states shown in Figure 153 (high-lighted in the main diagram - Figure 152). States PRES0 and PRES1 are concerned with detecting a picture clock via the signal presflg. Two cycles are allowed for the tests involved since they all depend on the value of rdyst, the adder output signal described in C.2.3.2.4. If a presentation flag is detected, all of the buffers are examined for possible 'readiness', otherwise the state machine just advances to state DRQ. Each cycle around the PRES0-PRES1 loop examines a different buffer, checking for full and ready conditions. If these are met, the previous ready buffer (if one exists) is cleared, the new ready buffer is allocated and its status is updated. This process is repeated until all buffers have been examined (index == max buf) and the state then advances. A buffer is deemed to be ready for display when any of the following is true:

```
(pic_num>pres_num)&&((pic_num - pres_num)>=128)

or

(pic_num<pres_num)&&((pres_num - pic_num)<=128)

or

pic_num == pres_num
```

State DRQ checks for a request for a display buffer (drq_valid_reg && disp_acc_reg). If there is no request the state advances (normally to state TOKEN - as will be described later). Otherwise, a display buffer index is issued as follows. If there is no ready buffer, the previous index is re-issued or, if there is no previous display buffer, a null index (zero) is issued. If a buffer state TOKEN to state PICTURE_END where, if the index is not already pointing at the current arrival buffer, it is set to point there so that its status can be updated. Assuming both out_acc_reg and en_full are true, status can be updated as described below. If not, control remains in state PICTURE_END until they are both true. The en_full signal is supplied by the write address generator to indicate that the swing buffer has swung, i.e., the last block has been successfully written and it is, therefore, safe to update the buffer status.

The just-completed buffer is tested for readiness and given the status either FULL or READY depending on the result of the test. If it is ready, rdy_buf is given the value of its index and the set_la_ev signal (late arrival event) is set high (indicating that the expected display has got ahead in time of the decoding). The new value of arr_buf now becomes zero and, if the previous ready buffer needs its status clearing, the index is set to point there and control moves to state VACATE_RDY. Otherwise, the index is reset to 1 and control returns to the start of the main loop.

C.2.4.6 Operation When PICTURE_START Received (Allocation of Arrival Buffer)

When a PICTURE_START token arrives during state TOKEN, the flag from_ps is set, causing the basic state machine loop to be changed such that state ALLOC is visited instead of state TOKEN. State ALLOC is concerned with allocating an arrival buffer (into which the arriving picture data can be written), and cycles through the buffers until it finds one whose status is VACANT. A buffer will only be allocated if out_acc_reg is high since it is output on the data two-wire interface. Accordingly, cycling around the loop will continue until this is indeed the case. Once a suitable arrival buffer has been found, the index is allocated to arr_buf and its status is flagged as IN_USE. Index is set to 1, the flag from_ps is reset, and the state is set to advance to NEW_EXP_TR. A check is made on the

The flag from_f1 is reset and the normal main loop operation is resumed.

C.2.4.8 Operation When TEMPORAL_REFERENCE Received

When a TEMPORAL_REFERENCE token is encountered, a check is made on the H.261 bit and, if set, the four states TEMP_REF0 to TEMP_REF3 are visited. These perform the following operations:

```
TEMP_REF0:temp_ref=in_data_reg;
TEMP_REF1:delta=temp_ref-exp_tr;index=arr_buf;
TEMP_REF2:exp_tr=delta+exp_tr;
TEMP_REF3:pic_num[i]=this_pnum+delta;index=1.
```

C.2.4.9 Other Tokens and Tails

State TOKEN passes control to state OUTPUT_TAIL in all cases other than those outlined above. Control remains here until the last word of the token is encountered (in_extn_reg is low) and the main loop is then re-entered.

C.2.5 Applications Notes

C.2.5.1 State Machine Stalling Buffer Manager Input

This requirement repeatedly check for the "asynchronous" timing events of picture clock and display buffer request. The necessity of having the buffer manager input stalled during these checks means that when there is a continuous supply of data at the input to the buffer manager, there will be a restriction on the data rate through the buffer manager. A typical sequence of states may be PRES0, PRES1, DRQ, TOKEN, OUTPUT_TAIL, each, with the exception of OUTPUT_TAIL, lasting one cycle. This means that for each block of 64 data items, there will be an overhead of 3 cycles during which the input is stalled (during states PRES0, PRES1 and DRQ) thereby slowing the write rate by 3/64 or approximately 5%. This number may occasionally increase to up to 13 cycles of overhead when auxiliary branches of the state machine are executed under worst-case conditions. Note that such large overheads will only apply on a once-per-frame basis.

C.2.5.2 Pres ntation Number Behavior During An Access

The particular embodiment of the bm_pres illustrated by

SECTION C.3 Write Address Generation

C.3.1 Introduction

The function of the write address generation hardware, in accordance with the present invention, is to produce block addresses for data to be written away to the buffers. This takes account of buffer base addresses, the component indicated in the stream, horizontal and vertical sampling within a macroblock, picture dimensions, and coding standard. Data arrives in macroblock form, but must be stored so that lines may be retrieved easily for display.

C.3.2 Functional Overview

Each time a new block arrives in the data stream (indicated by a DATA token), the write address generator is required to produce a new block address. It is not necessary to produce the address immediately, because up to 64 data words can be stored by the DRAM interface (in the swing buffer) before the address is actually needed. This means that the various address components can be added to a running total in successive cycles, and thus, hence obviating the need for any hardware multipliers. The macroblock counter function is effected by storing strategic terminal values and running counts in the register file, these being the operands for comparisons and conditional updates after each block address calculation.

Considering the picture format shown in Figure 161, expected address sequences can be derived for both standard and H.261-like data streams. These are shown below. Note that the format does not actually conform to the H.261 specification because the slices are not wide enough (3 macroblocks rather than 11) but the same "half-picture-width-slice" concept is used here for convenience and the sequence is assumed to be "H.261-type". Data arrives as full macroblocks, 4:2:0 in the example shown, and each component is stored in its own area of the specified buffer.

```
022,023,02E,02F,108,208;
036,037,042,043,10F,20F;
038,039,044,045,110,210;
03A,03B,046,047,111,211;
048,049,054,055,112,212;
04A,04B,056..........
........
06A,06B,076,077,11D,21D;
07E,07F,08A,08B,121,221;
080,081,08C,08D,122,222;
082,083,08E,08F,123;223;
```

C.3.3 Architecture

C.3.3.1 Interfaces

C.3.3.1.1 Interface to buffer manager

The buffer manager outputs data and the buffer index directly to the write address generator. This is performed under the control of a two-wire-interface. In some ways, it is possible to consider the write address generator block as an extension of the buffer manager because the two are very closely linked. They do, however, operate from two separate (but similar) clock generators.

C.3.3.1.2 Interface to dramif

The write address generator provides data and addresses for the DRAM interface. Each of these has their own two-wire-interface, and the dramif uses each of them in different clock regimes. In particular, the address is clocked into the dramif on a clock which is not related to the write address generator clock. It is, therefore, synchronized at the output.

C.3.3.1.3 Microprocessor Interface

The write address generator uses three bits of microprocessor address space together with 8-bit data bus and read and write strobes. There is a single select bit for register access.

indicates whether the state machine is in the middle of a two-cycle operation (i.e., an operation involving an add).

C.3.3.3.3. Component Count (wa comp)

Separate addresses are required for data blocks in each component, and this block maintains the current component under consideration based on the type of DATA header received in the input stream.

C.3.3.3.4 Modulo-3 Control (wa mod3)

When generating address sequences for H.261 data streams, it is necessary to count three rows of macroblocks to half way along the screen (see C.3.2). This is effected by maintaining a modulo-3 counter, incremented each time a new row of macroblocks is visited.

C.3.3.3.5 Control Registers (wa uregs)

Module wa_uregs contains the setup register and the coding standard register - the latter is loaded from the data stream. The setup register uses 3 bits: QCIF (lsb) and the maximum component expected in the data stream (bits 1 and 2). The access bit also resides in this block (synchronized as usual), with the "stopped" bits being derived at the next level up the hierarchy (walogic) as the OR of the access bit and the event stop bits. Microprocessor address decoding is done by the block wa_udec which takes read and write strobes, a select wire, and the lower two bits of the address bus.

C.3.3.3.6 Controlling State Machine (wa state)

The logic in this block is split into several distinct areas. The sate decode, new state encode, derivation of "intermediate" logic signals, datapath control signals (drivea, driveb, load, adder controls and select signals), multiplexer controls, two-wire-interface controls, and the five event signals.

C.3.3.3.7 Event Generation

The five event bits are generated as a result of certain tokens arriving at the input. It is important that, in each case, the entire token is received before any events are generated because the event service routines perform

Keyhole Register Name	Keyhole Address	Bits	Comments
BU_WADDR_BUFFER0_BASE_MSB	0x85	2	Must be Loaded
BU_WADDR_BUFFER0_BASE_MID	0x86	8	
BU_WADDR_BUFFER0_BASE_LSB	0x87	8	
BU_WADDR_BUFFER1_BASE_MSB	0x89	2	Must be Loaded
BU_WADDR_BUFFER1_BASE_MID	0x8a	8	
BU_WADDR_BUFFER1_BASE_LSB	0x8b	8	
BU_WADDR_BUFFER2_BASE_MSB	0x8d	2	Must be Loaded
BU_WADDR_BUFFER2_BASE_MID	0x8e	8	
BU_WADDR_BUFFER2_BASE_LSB	0x8f	8	
BU_WADDR_COMP0_HMBADDR_MSB	0x91	2	Test only
BU_WADDR_COMP0_HMBADDR_MID	0x92	8	
BU_WADDR_COMP0_HMBADDR_LSB	0x93	8	
BU_WADDR_COMP1_HMBADDR_MSB	0x95	2	Test only
BU_WADDR_COMP1_HMBADDR_MID	0x96	8	
BU_WADDR_COMP1_HMBADDR_LSB	0x97	8	
BU_WADDR_COMP2_HMBADDR_MSB	0x99	2	Test only
BU_WADDR_COMP2_HMBADDR_MID	0x9a	8	
BU_WADDR_COMP2_HMBADDR_LSB	0x9b	8	
BU_WADDR_COMP0_VMBADDR_MSB	0x9d	2	Test only
BU_WADDR_COMP0_VMBADDR_MID	0x9e	8	
BU_WADDR_COMP0_VMBADDR_LSB	0x9f	8	
BU_WADDR_COMP1_VMBADDR_MSB	0xa1	2	Test only
BU_WADDR_COMP1_VMBADDR_MID	0xa2	8	
BU_WADDR_COMP1_VMBADDR_LSB	0xa3	8	
BU_WADDR_COMP2_VMBADDR_MSB	0xa5	2	Test only
BU_WADDR_COMP2_VMBADDR_MID	0xa6	8	
BU_WADDR_COMP2_VMBADDR_LSB	0xa7	8	
BU_WADDR_VBADDR_MSB	0xa9	2	Test only
BU_WADDR_VBADDR_MID	0xaa	8	
BU_WADDR_VBADDR_LSB	0xab	8	

Tabl C.3.2 Image Formatter Address Generator Keyhole

Keyhole Register Name	Keyhole Address	Bits	Comments
BU_WADDR_COMP0_LAST_MB_IN_ROW_MSB	0xd5	2	Must be Loaded
BU_WADDR_COMP0_LAST_MB_IN_ROW_MID	0xd6	8	
BU_WADDR_COMP0_LAST_MB_IN_ROW_LSB	0xd7	8	
BU_WADDR_COMP1_LAST_MB_IN_ROW_MSB	0xd9	2	Must be Loaded
BU_WADDR_COMP1_LAST_MB_IN_ROW_MID	0xda	8	
BU_WADDR_COMP1_LAST_MB_IN_ROW_LSB	0xdb	8	
BU_WADDR_COMP2_LAST_MB_IN_ROW_MSB	0xdd	2	Must be Loaded
BU_WADDR_COMP2_LAST_MB_IN_ROW_MID	0xde	8	
BU_WADDR_COMP2_LAST_MB_IN_ROW_LSB	0xdf	8	
BU_WADDR_COMP0_LAST_MB_IN_HALF_ROW_MSB	0xe1	2	Must be Loaded
BU_WADDR_COMP0_LAST_MB_IN_HALF_ROW_MID	0xe2	8	
BU_WADDR_COMP0_LAST_MB_IN_HALF_ROW_LSB	0xe3	8	
BU_WADDR_COMP1_LAST_MB_IN_HALF_ROW_MSB	0xe5	2	Must be Loaded
BU_WADDR_COMP1_LAST_MB_IN_HALF_ROW_MID	0xe6	8	
BU_WADDR_COMP1_LAST_MB_IN_HALF_ROW_LSB	0xe7	8	
BU_WADDR_COMP2_LAST_MB_IN_HALF_ROW_MSB	0xe9	2	Must be Loaded
BU_WADDR_COMP2_LAST_MB_IN_HALF_ROW_MID	0xea	8	
BU_WADDR_COMP2_LAST_MB_IN_HALF_ROW_LSB	0xeb	8	
BU_WADDR_COMP0_LAST_ROW_IN_MB_MSB	0xed	2	Must be Loaded
BU_WADDR_COMP0_LAST_ROW_IN_MB_MID	0xee	8	
BU_WADDR_COMP0_LAST_ROW_IN_MB_LSB	0xef	8	
BU_WADDR_COMP1_LAST_ROW_IN_MB_MSB	0xf1	2	Must be Loaded
BU_WADDR_COMP1_LAST_ROW_IN_MB_MID	0xf2	8	
BU_WADDR_COMP1_LAST_ROW_IN_MB_LSB	0xf3	8	
BU_WADDR_COMP2_LAST_ROW_IN_MB_MSB	0xf5	2	Must be Loaded
BU_WADDR_COMP2_LAST_ROW_IN_MB_MID	0xf6	8	
BU_WADDR_COMP2_LAST_ROW_IN_MB_LSB	0xf7	8	
BU_WADDR_COMP0_BLOCKS_PER_MB_ROW_MSB	0xf9	2	Must be Loaded
BU_WADDR_COMP0_BLOCKS_PER_MB_ROW_MID	0xfa	8	
BU_WADDR_COMP0_BLOCKS_PER_MB_ROW_LSB	0xfb	8	
BU_WADDR_COMP1_BLOCKS_PER_MB_ROW_MSB	0xfd	2	Must be Loaded
BU_WADDR_COMP1_BLOCKS_PER_MB_ROW_MID	0xfe	8	
BU_WADDR_COMP1_BLOCKS_PER_MB_ROW_LSB	0xff	8	
BU_WADDR_COMP2_BLOCKS_PER_MB_ROW_MSB	0x101	2	Must be Loaded
BU_WADDR_COMP2_BLOCKS_PER_MB_ROW_MID	0x102	8	
BU_WADDR_COMP2_BLOCKS_PER_MB_ROW_LSB	0x103	8	

Table C.3.2 Image formatt r Addr ss Generat r Keyhol

C.3.4 Programming the Write Address Gen rator

The following datapath registers must contain the correct picture size information before address calculation can proceed. They are illustrated in Figure 162.

1) WADDR_HALF_WIDTH_IN_BLOCKS: this defines the half width, in blocks, of the incoming picture.
2) WADDR_MBS_WIDE: this defines the width, in macroblocks, of the incoming picture.
3) WADDR_MBS_HIGH: this defines the height, in macroblocks, of the incoming picture.
4) WADDR_LAST_MB_IN_ROW: this defines the block number of the top left hand block of the last macroblock in a single, full-width row of macroblocks. block numbering starts at zero in the top left corner of the left-most macroblock, increases across the frame with each block and subsequently with each following row of blocks within the macroblock row.
5) WADDR_LAST_MB_IN_HALF_ROW: this is similar to the previous item, but defines the block number of the top left block in the last macroblock in a half-width row of macroblocks.
6) WADDR_LAST_ROW_IN_MB: this defines the block number of the left most block in the last row of blocks within a row of macroblocks.
7) WADDR_BLOCKS_PER_MB_ROW: this defines the total number of blocks contained in a single, full-width row of macroblocks.
8) WADDR_LAST_MB_ROW: this defines the top left block address of the left-most macroblock in the last row of macroblocks in the picture.
9) WADDR_HBS: this defines the width in blocks of the incoming picture.
10) WADDR_MAXHB: this defines the block number

C.3.5 Operation of Th State Machine

There are 19 states in the buffer manager's state machine, as detailed in Table C.3.3. These interact as shown in Figure 164, and also as described in the behavioral description, bmlogic.M.

State	Value
IDLE	0x00
DATA	0x10
CODING_STANDARD	0x0C
HORZ_MBS0	0x07
HORZ_MBS1	0x06
VERT_MBS0	0x0B
VERT_MBS1	0x0A
OUTPUT_TAIL	0x08
HB	0x11
MB0	0x1D
MB1	0x12
MB2	0x1E
MB3	0x13
MB4	0x0E
MB5	0x14
MB6	0x15
MB4A	0x18

Table C.3.3 Write Address G n rator States

2) BU_WADDR_HMBADDR: the block address (top left block) of the top macroblock of the column of macroblocks in which the block whose address is being calculated is contained.
3) BU_WADDR_VBADDR: the block address, *within the macroblock row*, of the left-most block of the row of blocks in which the block whose address is being calculated is contained.
4) BU_WADDR_HB: the horizontal block number, within the macroblock, of the block whose address is being calculated.
5) BU_WADDR_SCRATCH: the scratch register used for temporary storage of intermediate results.

Considering Figure 163, and taking, for example, the calculation of the block whose address is 0x62D, the following sequence of calculations will take place;

```
SCRATCH=BUFFERn_BASE+COMPm_OFFSET; (assume 0)
SCRATCH=0+0x5D8;
SCRATCH=0x5D8+0x28;
SCRATCH=0x600+0x2C;
block address=0x62C+1=0x62D;
```

The contents of the various registers are illustrated in the Figure.

C.3.5.2 Calculation of New Screen Location Parameters

When the address has been output, the state machine continues to perform calculations in order to update the various screen location parameters described above. The states HB and MB0 through to MB6 do the calculations, transferring control at some point to state DATA from which the reminder of the DATA Token is output.

These states proceed in pairs, the first of a pair calculating the difference between the current count and its terminal value and, hence, generating a zero flag. The second of the pair either resets the register or adds a fixed (based on values in the setup registers derived from screen size) offset. In ach case, if the count under consideration has reached its terminal value (i.e., the

C.3.5.2.1 Calculations for Standard (MPEG-styl) Sequenc s

The sequence of operations is as follows (in which the zero flag is based on the output of the adder):

```
states HB and MBO:
scratch = hb - maxhb;
if (z)
  hb = 0;
else
(
  hb = hb + 1
  new_state = DATA;
)
states MB1 and MB2:
scratch = vb_addr - last_row_in_mb;
if (z)
  vb_addr = 0;
else
(
  vb_addr = vb_addr + width_in_blocks;
  new_state = DATA;
)
states MB3 and MB4:
scratch = hmb_addr - last_mb_in_row;
if (z)
  hmb_addr = 0;
else
(
  hmb_addr = hmb_addr + maxhb;
  new_state = DATA;
)
states MB5 and MB6:
scratch = vmb_addr - last_mb_row;
if (!z)
  vmb_addr = vmb_addr + blocks_per_mb_row;
```

(vmb_addr is reset after a PICTURE_START token is detected, rather than when the end of a picture is inferred state MB4A:

```
vmb_addr = vmb_addr + blocks_per_mb_row;
new_state = DATA;
```

state (MB4) and MB4B:

```
(scratch = hmb_addr - last_mb_in_half_row;)
if (z & (mod3==2)) /*end of slice on left of screen*/
{
   hmb_addr = hmb_addr + maxhb;
   new_state = MB4C;
}
else if (z) /*end of row on left of screen*/
{
   hmb_addr = 0;
   new_state = MB4A;
}
else
{
   hmb_addr = hmb_addr + maxhb;
   new_state = DATA;
}
```

states MB4C and MB4D:

```
vmb_addr = vmb_addr - blocks_per_mb_row;
vmb_addr = vmb_addr - blocks_per_mb_row;
new_state = DATA;
```

states MB5and MB6:- as above

C.3.5.3 Operation on PICTURE_START Token

When a PICTURE_START token is received, control passes to state PIC_ST1 where the vb_addr register (BU_WADDR_VBADDR) is reset to 0. Each of states PIC_ST2 and PIC_ST3 are then visited, once for each component, resetting hmb_addr and vmb_addr respectively. Control then returns, via state OUTPUT_TAIL, to IDLE.

SECTION C.4 Read Address Generator

C.4.1 Overvi w

The read address generator of the present invention consists of four state machine/datapath blocks. The first, "dline", generates line addresses and distributes them to the other three (one for each component) identical page/block address generators, "dramctls". All blocks are linked by two wire interfaces. The modes of operation include all combinations of interlaced/progressive, first field upper/lower, and frame start on upper/lower/both. The Table C.3.4 shows the names, addresses, and reset states of the dispaddr control registers, and Chapter C.13 gives a programming example for both address generators.

C.4.2 Line Address Generator (dline)

This block calculates the line start addresses for each component. Table C.3.4 shows the 18 bit datapath registers in dline.

Note the distinction between DISP_register_name and ADDR_register_name DISP _name registers are in dispaddr only and means that the register is specific to the display area to be read out of the DRAM. ADDR_name means that the register describes something about the structure of the external buffers.

Operation

The basic operation of dline, ignoring all modes repeats etc. is:

```
if (vsync_start)/* first active cycle of vsync*/
(
comp = 0
DISP_VB_CNT_COMP[comp]=0;
LINE[comp]=BUFFER_BASE[comp]+0;
LINE[comp]=LINE[comp]+DISP_COMP_OFFSET[comp];
while (VB_CNT_COMP[comp]<DISP_VBS_COMP[comp]
(
while (line_count[comp]<8)
(
```

Register Names	Bus	Keynole Address	Description	Comments
BUFFER_BASE0	A	0x00,01,02,03	Block address of the start of each buffer.	These registers must be loaded by the upi before operation can begin.
BUFFER_BASE1	A	0x04,05,06,07		
BUFFER_BASE2	A	0x08,09,0a,0b		
DISP_COMP_OFFSET0	B	0x24,25,26,27	Offsets from the buffer base to where reading begins.	
DISP_COMP_OFFSET1	B	0x28,29,2a,2b		
DISP_COMP_OFFSET2	B	0x2c,2d,2e,2f		
DISP_VBS_COMP0	B	0x30,31,32,33	Number of vertical blocks to be read	
DISP_VBS_COMP1	B	0x34,35,36,37		
DISP_VBS_COMP2	B	0x38,39,3a,3b		
ADDR_HBS_COMP0	B	0x3c,3d,3e,3f	Number of horizontal blocks IN THE DATA	
ADDR_HBS_COMP1	B	0x40,41,42,43		
ADDR_HBS_COMP2	B	0x44,45,46,4		
LINE0	A	0x0c,0d,0e,0f	Current line address	These registers are temporary locations used by dispaddr. Note: All registers are R/W from the upi
LINE1	A	0c10,11,12,13		
LINE2	A	0x14,15,16,17		
DISP_VB_CNT_COMP0	A	0x18,19,1a,1b	Number of vertical blocks remaining to be read.	
DISP_VB_CNT_COMP1	A	0x1c,1d,1e,1f		
DISP_VB_CNT_COMP2	A	0x20,21,22,23		

Table C.3.4 Dispaddr Datapath R gisters

Note that the upi is actively locked out from the datapath registers until the access bit is "1". In order for access to dispaddr to be achieved without disrupting the current display or datapath operation, access will only given and released under the following circumstances.

Stopping: Access will only be granted if the datapath has finished its current two cycle operation (if it were doing one), and the "safe" signal from the output controller is high. This signal represents the area on the screen below the display window and is programmed in the output controller (not dispaddr). Note: It is, therefore, necessary to program the output controller before trying to gain access to dispaddr.

Starting-Access will only be released when "safe" is high, or during vsync. This ensures that display will not start too close to the active window.

This scheme allows the controlling software to request access, poll until end of display, modify dispaddr, and release access. If the software is too slow and doesn't release the access bit until after vsync, dispaddr will not start until the next safe period. Border color will be displayed during this "lost" picture (rather than rubbish).

C.4.3.3 DISPADDR_CTLO[7:0]

When reading the following descriptions, it is important to understand the distinction between interlaced data and an interlaced display.

Interlaced data can be of two forms. The Top-Level Registers supports field-pictures (each buffer contains one field), and frames (each buffer contains an entire frame - interlaced or not)

DISPADDR_CTL0[7:0]contains the following control bits:

SYNC_MODE[1:0]

With an interlaced display, vsyncs referring to top and bottom fields are differentiated by the field_info pin. In this context, field_info = HIGH meaning the top field. These two control bits determine which vsyncs dispaddr will request a new display buffer from the buffer manager and, LINE_RPT[2:0]

Each bit, when set, causes the lines of the corresponding component to be read twice (bit 0 affects component 0 etc.). This forms the first part of the vertical unsampling. It is used in the 8 times chroma upsampling required for conversion from QFIF to 601.

COMPOHOLD

This bit is used to program the ratio of the number of lines to be read (as opposed to displayed) for component 0 to those of components 1 and 2).

0: Same number of lines, i.e., 4:4:4 data in the buffers.

1: Twice as many component 0 lines, i.e., 4:2:0.

Page/Block Address Generators (dramctls)

When passed a line address, these blocks generate a series of page/line addresses and blocks to read along the line. The minimum page width of 8 blocks is always assumed and the resulting outputs consist of a page address, a 3 bit line number, a 3 bit block start, and a 3 bit block stop address. (The line number is calculated by dline and passed through the dramctls unmodified). Thus, to read out 48 pixels of line 5 form page Oxaa starting from the third block from the left (an arbitrary point along an arbitrary line), the addresses passed to the DRAM interface would be:

Page	= Oxaa
Line	= 5
Block start	= 2
Block stop	= 7

Each of these three machines has 5 datapath registers. These are shown in Table C.3.4. The basic behavior of each dramctl is:

Table C.3.5 Dramctl(0,1 &2) Datapath R gisters

Table C.3.5 Dramctl(0,1 & 2) Datapath Registers

Register Names	Bus	Keyhole Address	Description	Comments
DISP_COMP0_HBS	A	0x48,49,4a,4b	The number of horizontal blocks to be read. c.f. ADDR_HBS	This register must be loaded before operation can begin.
DISP_COMP1_HBS	A	0x4c,4d,4e,4f		
DISP_COMP2_HBS	A	0x50,51,52,53		
CNT_LEFT0	A	0x54,55,56,57	Number of blocks remaining to be read	These registers are temporary locations used by dispaddr. Note: All registers are R/W from the upi
CNT_LEFT1	A	0x58,59,5a,5b		
CNT_LEFT2	A	0x5c,5d,5e,5f		
PAGE_ADDR0	A	0x60,61,62,63	The address of the current page.	
PAGE_ADDR1	A	0x64,65,66,67		
PAGE_ADDR2	A	0x68,69,6a,6b		
BLOCK_ADDR0	B	0x6c,6d,6e,6f	Current block address	
BLOCK_ADDR1	B	0x70,71,72,73		
BLOCK_ADDR2	B	0x74,75,76,77		
BLOCKS_LEFT0	B	0x78,79,7a,7b	Blocks left in current page	
BLOCKS_LEFT1	B	0x7c,7d,7e,7f		
BLOCKS_LEFT2	B	0x80,81,82,83		

Programming

The following 15 dispaddr registers must be programmed before operation can begin.

BUFFER_BASE0,1,2

DISP_COMP_OFFSET0,1,2

DISP_VBS_COMP0,1,2

ADDR_HBS_COMP0,1,2

DISP_COMP0,1,2_HBS

SECTION C.5 Datapaths for Address Generation

The datapaths used in dispaddr and waddrgen are identical in structure and width (18 bits), only differing in the number of registers, some masking, and the flags returned to the state machine. The circuit of one slice is shown in Figure 165, "Slice Of Datapath,". Registers are uniquely assigned to drive the A or B bus and their use (assignment) is optimized in the controller. All registers are loadable from the C bus, however, not all "load" signals are driven. All operations involving the adder cover two cycles allowing the adder to have ordinary ripple carry. Figure 166, "Two cycle operation of the datapath," shows the timing for the two cycle sum of two registers being loaded back into the "A" bus register. The various flags are "phO"ed within the datapath to allow ccode generation. For the same reason, the structure of the datapath schematics is a little unusual. The tristates for all the registers (onto the A and B buses) are in a single block which eliminates the combinatorial path in the cell, therefore, allowing better ccode generation. To gain upi access to the datapaths, the access bit must be set, for without this, the upi is locked out. Upi access is different from read and write:

- •Writing: When the access bit is set, all load signals are disabled and one of a set of three byte addressed write strobes driven to the appropriate byte of one of the registers. The upi data bus passes vertically down the datapath (replicated, 2-8-8 bits) and the 18 bit register is written as three separate byte writes
- •Reading: This is achieved using the A and B buses. Once again, the access bit must be set. The addressed register is driven onto the A or B bus and a upi byte select picks a byte from the relevant bus and drives it onto th upi bus.

As double cycle datapath operations require the A and B buses to retain their values, and upi accesses disrupt

SECTION C.6 The DRAM Interface

C.6.1 Overvi w

In the present invention, the Spacial Decoder, Temporal Decoder and Video Formatter each contain a DRAM Interface block for that particular chip. In all three devices, the function of the DRAM Interface is to transfer data from the chip to the external DRAM and from the external DRAM into the chip via block addresses supplied by an address generator.

The DRAM Interface typically operates from a clock which is asynchronous to both the address generator and to the clocks of the various blocks through which data is passed. This asynchronism is readily managed, however, because the clocks are operating at approximately the same frequency.

Data is usually transferred between the DRAM Interface and the rest of the chip in blocks of 64 bytes (the only exception being prediction data in the Temporal Decoder). Transfers take place by means of a device known as a "swing buffer". This is essentially a pair of RAMs operated in a double-buffered configuration, with the DRAM interface filling or emptying one RAM while another part of the chip empties or fills the other RAM. A separate bus which carries an address from an address generator is associated with each swing buffer.

Each of the chips has four swing buffers, but the function of these swing buffers is different in each case. In the Spacial Decoder, one swing buffer is used to transfer coded data to the DRAM, another to read coded data from the DRAM, the third to transfer tokenized data to the DRAM and the fourth to read tokenized data from the DRAM. In the Temporal Decoder, one swing buffer is used to write Intra or Predicted picture data to the DRAM, the second to read Intra or Predicted data from the DRAM and the other two to read forward and backward prediction data. In the Video Formatter, one swing buffer is used to transfer data to the DRAM and the other three are used to read data from

SECTION C.7 Vertical Upsampling

C.7.1 Introduction

Given a raster scan of pixels of one color component at its input, the vertical upsampler in accordance with the present invention, can provide an output scan of twice the height. Mode selection allows the output pixel values to be formed in a number of ways.

C.7.2 Ports

Input two wire interface:

- in_valid
- in_accept
- in_data[7:0]
- in_lastpel
- in_lastline

Output two wire interface:

- out_valid
- out_accept
- out_data[9:0]
- out_last

mode[2:0]

nupdata[7:0], upaddr, upsel[3:0], uprstr, upwstr ramtest

tdin, tdout, tph0, tckm, tcks

ph0, ph1, notrst0

C.7.3 Mode

As selected by the input bus mode[2:0].

Mode register values 1 and 7 are not used.

In each of the above modes, the output pixels are represented as 10-bit values, not as bytes. No rounding or truncation takes place in this block. Where necessary, values are shifted left to use the same range.

C.7.3.1 Mode 0:Fifo

The block simply acts as a FIFO store. The number of output pixels is exactly the same as at the input. The values are shifted left by two.

length of the line in "FIFO" mode.

The input signals in_lastpel and in_lastline are used to indicate the end of the input line and the end of the picture. In_lastpel, it should be high coincident with the last pixel of each line. In_lastline, it should be high coincident with the last pixel of the last line of the picture.

The output signal out_last is high coincident with the last pixel of each output line.

In "repeat" mode, each line is written into store "a". The line is then read out twice. As it is read out for the second time, the next line may start to be written.

In "lower", "upper" and "central" modes, lines are written alternately into stores "a" and "b". The first line of a picture is always written into store "a". Two tiny state machines, one for each store, keep track of what is in each store and which output line is being formed. From these states are generated the read and write requests to the linestore RAMs, and the signals that determine when the next line may overwrite the present data.

A register (lastaddr) stores the write address when in_lastpel is high, thereby providing the length of the line for the formation of the output lines.

C.7.5 UPI

This block contains two 512 x 8 bit RAM arrays, which may be accessed via the microprocessor interface in the typical way. There are no registers with microprocessor access.

Table C.7.2 Coeffici nts for Mode 1

Coeff	All clock periods
k0	c00
k1	c10
k2	c20

Table C.7.3 Coefficients for Mode 2

Coeff	1st clock period	2nd clock period
k0	c00	c01
k1	c10	c11
k2	c20	c21

Table C.7.4 Coefficients for Mode 3

Coeff	1st clock period	2nd clock period	3rd clock period	4th clock period
k0	c00	c01	c02	c03
k1	c10	c11	c12	c13
k2	c20	c21	c22	c23

C.8.3 Description f a Horizontal Up-Sampler

The datapath of the Horizontal Up-sampler is illustrated in Figure 168.

The operation is outlined below for the x4 upsample case. In addition, x2 upsampling and x1 filtering (modes 2 and 1) are degenerate cases of this, and bypass (mode 0) the entire filter, data passing straight from the input latch to the output latch via the final mux, as illustrated.

1) When valid data is latched in the input latch ("L"), it is held for 4 clock periods.

2) The coefficient registers (labelled "COEFF") are multiplexed onto the multipliers for one clock period, each in turn, at the same time as the two sets of four pipeline registers (labelled "PIPE") are clocked. Thus, for input data x_n, the first PIPE will fill up with the values $c00.x_n$, $c01.x_n$, $c02.x_n$, $c03.x_n$.

3) Similarly, the second multiplier will multiply x_n by of its coefficients, in turn, and the third multiplier by all its coefficients, in turn.

It can be seen that the output will be of the form shown in Table C.7.6

Table C.7.6 Output Sequence for Mode 3

Clockl Period	Output
0	$c20.x_n + c10.x_{n-1} + c00.x_{n-2}$
1	$c21.x_n + c11.x_{n-1} + c01.x_{n-2}$
2	$c22.x_n + c12.x_{n-1} + c02.x_{n-2}$
3	$c23.x_n + c13.x_{n-1} + c03.x_{n-2}$

From the point of view of the output, each clock period produces an individual pixel. Since each output pixel is dependent on the weighted values of 12 input pixels (although there are only three different values), this can replicated twice-over and the clearing down of the pipeline between lines (the pipeline contains partially-processed redundant data immediately after a line has been completed).

quantities are in the range of (32..470). Since the input to the **Top-Level Registers** CSC is Y, C_R, C_B, only the third and fourth of these equations are of relevance.

In the CSC design, the precision of the coefficients was chosen so that, for 9 bit data, all output values were within plus or minus 1 bit of the values produced by a full floating point simulation of the algorithm (this is the best accuracy that it is possible to achieve). This gave 13 bit twos-complement coefficients for cx0-cx3 and 14 bit twos-complement coefficients for cx4. The coefficients for all the design conversions are given below in both decimal and hex.

Table C.8.1 Coefficients for Various Conversions

	E_R->Y		R->Y		Y->E_R		Y->R	
Coeff	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex
c01	0.299	0132	0.256		1.0	0400	1.169	04AD
c02	0.587	0259	0.502		1.402	059C	1.639	068E
c03	0.114	0075	0.098		0.0	0000	0.0	0000
c04	0.0	0000	16		-179.456	F4C8	-223.478	F1B3
c11	0.5	0200	0.428		1.0	0400	1.169	04AD
c12	-0.419	FE53	-0.358		-0.714	FD25	-0.835	FCA9
c13	-0.081	FFAD	-0.070		-0.344	FEA0	-0.402	FE64
c14	128.0	0800	128		135.5	0878	139.7	08BA
c21	-0.169	FF53	-0.144		1.0	0400	1.169	04AD
c22	-0.331	FEAD	-0.283		0.0	0000	0.0	0000
c23	0.5	0200	0.427		1.772	0717	2.071	0849
c24	128	0800	128		-226.816	F1D2	-283.84	EE42

All these numbers are calculated from the fundamental equation:

$$Y = 0.299E_R+0.587E_G+0.0114E_B$$

and the following color-difference equations:

$$C_R=E_R-Y$$

$$C_B=E_B-Y$$

C.9.3 Description f the CSC

The structure of the CSC is illustrated in Figure 169, where only two of the three "components" have been shown because of space limitations. In the Figure, "register" or "R" implies a master-slave register and "latch" or "L" implies a transparent latch.

All coefficients are loaded into read-write UPI registers which are not shown explicitly in the Figure. To understand the operation, consider the following sequence with reference to the left-most "component" (that which produces output out_data0):

1)Data arrives at inputs x0-2 (in_data0-2). This represents a single pixel in the input color-space. This is latched.

2)x0 is multiplied by c01 and latched into the first pipeline register. x1 and x2 move on one register.

3)x1 is multiplied by c02, added to (x1.c01) and latched into the next pipeline register. x2 moves on one register.

4)x2 is multiplied by c03 and added to the result of (3), producing (x1.c01 + x2.c02 + x3.c03). The result is latched into the next pipeline register.

5)The result of (4) is added to c04. Since data is kept in carry-save format through the multipliers, this adder is also used to resolve the data from the multiplier chain. The result is latched in the next pipeline register.

6)The final operation is to saturate the data. Partial results are passed from the resolving adder to the saturate block to achieve this.

It can be seen that the result is y0, as specified in the matrix equation at the start of this section. Similarly, y1 and y2 are formed in the same manner.

Three multipliers are used, with the coefficients as the multiplicand and the data as the multiplicator. This allows an efficient layout to be achieved, with partial results flowing down the datapath and the same input data

SECTION C.10 Output Controller

C.10.1 Introduction

The output controller, in accordance with the present invention, handles the following functions:

- It provides data in one of three modes
  - 24-bit 4:4:4
  - 16-bit 4:2:2
  - 8-bit 4:2:2
- It aligns the data to the video display window defined by the vsync and hsync pulses and by programmed timing registers
- It adds a border around the video window, if required

C.10.2 Ports

Input two wire interface:

- in_valid
- in_accept
- in_data[23:0]

Output two wire interface:

- out_valid
- out_accept
- out_data[23:0]
- out_active
- out_window
- out_comp[1:0]

in_vsync, in_hsync
nupdata[7:0], upaddr[4:0], upsel, rstr, wstr
tdin, tdout, tph0, tckm, tcks chiptest
ph0, ph1, notrst0, notrst1

C.10.3 Out Modes

The format of the output is selected by writing to the opmode register.

C.10.3.1 Mode 0

This mode is 24-bit 4:4:4 RGB or YCrCb. Input data passes directly to the output.

in_data [7:0].

C.10.4 Output Flags

- •out_active indicates that the output data is part of the active window, i.e., video data or border.
- •out_window indicates that the output data is part of the video window.
- •out_comp[1:0] indicates which color component is present on out_data[7:0] in output modes 1 and 2. In mode 1, 0=Cb, 1=Cr. In mode 2, 0=Y, 1=Cr, 2=Cb.

C.10.5 Two-Wire Mode

The two-wire mode of the present invention is selected by writing 1 to the two wire register. It is not selected following reset. In two wire mode, the output timing registers and sync signals are ignored and the flow of data through the block is controlled by out_accept. Note that in normal operation, out_accept should be tied high.

C.10.6 Snooper

There is a super-snooper on the output of the block which includes access to the output flags.

C.10.7 How It Works

Two identical down-counters keep track of the current position in the display. "Vcount" decrements on hsyncs and loads from the appropriate timing register on vsync or at its terminal count. "Hcount" decrements on every pixel and loads on hsync or at its terminal count. Note that in output mode 2, one pixel corresponds to two clock cycles.

the divisor has also been altered) will merely cause the Clock Divider to lock to its new frequency "on-the-fly." Once activated, there is no way of halting the Clock Divider other than by Chip RESET.

Table C.10.1 Clock Divider Registers

Address	Register
00b	access bit
01b	divisor MSB
10b	divisor
11b	divisor LSB

Any divisor value in the range 16 to 16,777,216 may be used.

C.11.3 Description of the Clock Divider

The Clock Divider is implemented as four 22 bit counters which are cascaded such that as one counter carries, it will activate the next counter in turn. A counter will count down the value of divisor/4 before carrying and, therefore, each counter will take it, in turn, to generate a pulse of the divided clock frequency.

After carrying, the counter will reload with divisor/8 and this is counted down to produce the approximate equal mark-space ratio divided clock. As each counter reloads from the divisor register when it is activated by the previous counter, this enables the divided clock frequency to be changed on the fly by simply altering the contents of the divisor.

Each counter is clocked by its own independent clock generator in order to control clock skew between counters precisely and to allow each counter to be clocked by a different set of clocks.

A state machine controls the generation of the divisor/4 and divisor/8 values and also multiplexes the correct source clocks from the PLL to the clock generators. The

SECTION C.12 Address Maps

C.12.1 Top Level Address Map

Notes:-

1)The register for the Top Level Address Map as set forth in Table C.11.1 are the names used during the design. They are not necessarily the names that will appear on the datasheet.

2)Since this is a full address map, many of the locations listed here include locations for test only.

REGISTER NAME	Address	Bits	COMMENT
BU_EVENT	0x0	8	Write 1 to reset
BU_MASK	0x1	8	R/W
BU_EN_INTERRUPTS	0x2	1	R/W
BU_WADDR_COD_STD	0x4	2	R/W
BU_WADDR_ACCESS	0x5	1	R/W- access
BU_WADDR_CTL1	0x6	3	R/W
BU_DISPADDR_LINES_IN_LAST_ROW0	0x8	3	R/W
BU_DISPADDR_LINES_IN_LAST_ROW1	0x9	3	R/W
BU_DISPADDR_LINES_IN_LAST_ROW2	0xa	3	R/W
BU_DISPADDR_ACCESS	0xb	1	R/W- access
BU_DISPADDR_CTL0	0xc	8	R/W
BU_DISPADDR_CTL1	0xd	1	R/W
BU_BM_ACCESS	0x10	4	R/W- access
BU_BM_CTL0	0x11	2	R/W
BU_BM_TARGET_IX	0x12	4	R/W
BU_BM_PRES_NUM	0x13	8	R/W-asynchronous
BU_BM_THIS_PNUM	0x14	8	R/W
BU_BM_PIC_NUM0	0x15	8	R/W
BU_BM_PIC_NUM1	0x16	8	R/W
BU_BM_PIC_NUM2	0x17	8	R/W
BU_BM_TEMP_REF	0x18	5	RO

Tabl C.11.1 Top-Level Registers A Top Level Address Map

REGISTER NAME	Address	Bits	COMMENT
BU_IF_CONFIGURE	0x60	5	R/W
BU_UV_MODE	0x61	6	R/W- xnnnxnnn
BU_COEFF_KEYADDR	0x62	7	R/W - See Table C.11.3
BU_COEFF_KEYDATA	0x63	8	for contents.
BU_GA_ACCESS	0x68	1	R/W
BU_GA_BYPASS	0x69	1	R/W
BU_GA_RAM0_ADDR	0x6a	8	R/W
BU_GA_RAM0_DATA	0x6b	8	R/W
BU_GA_RAM1_ADDR	0x6c	8	R/W
BU_GA_RAM1_DATA	0x6d	8	R/W
BU_GA_RAM2_ADDR	0x6e	8	R/W
BU_GA_RAM2_DATA	0x6f	8	R/W
BU_DIVA_3	0x70	1	R/W
BU_DIVA_2	0x71	8	R/W
BU_DIVA_1	0x72	8	R/W
BU_DIVA_0	0x73	8	R/W
BU_DIVP_3	0x74	1	R/W
BU_DIVP_2	0x75	8	R/W
BU_DIVP_1	0x76	8	R/W
BU_DIVP_0	0x77	8	R/W
BU_PAD_CONFIG_1	0x78	7	R/W
BU_PAD_CONFIG_0	0x79	8	R/W
BU_PLL_RESISTORS	0x7a	8	R/W
BU_REF_INTERVAL	0x7b	8	R/W
BU_REVISION	0xff	8	RO- revision
The following registers are in the "test space". They are unlikely to appear on the datasheet.			
BU_BM_PRES_FLAG	0x80	1	R/W
BU_BM_EXP_TR	0x81	**	These registers are
BU_BM_TR_DELTA	0x82	**	missing on revA
BU_BM_ARR_IX	0x83	2	R/W
BU_BM_DSP_IX	0x84	2	R/W
BU_BM_RDY_IX	0x85	2	R/W
BU_BM_BSTATE3	0x86	2	R/W
BU_BM_BSTATE2	0x87	2	R/W

Table C.11.1 Top-L vel R gist rs A Top L vel Address Map (contd)

REGISTER NAME	Address	Bits	COMMENT
BU_IF_SNP0_1	0xb8	8	R/W - Three snoopers on the dramif data outputs.
BU_IF_SNP0_0	0xb9	8	
BU_IF_SNP1_1	0xba	8	
BU_IF_SNP1_0	0xbb	8	
BU_IF_SNP2_1	0xbc	8	
BU_IF_SNP2_0	0xbd	8	
BU_IFRAM_ADDR_1	0xc0	1	R/W - upi access to IF RAM
BU_IFRAM_ADDR_0	0xc1	8	
BU_IFRAM_DATA	0xc2	8	
BU_OC_SNP_3	0xc4	8	R/W - snooper on output of chip
BU_OC_SNP_2	0xc5	8	
BU_OC_SNP_1	0xc6	8	
BU_OC_SNP_0	0xc7	8	
BU_YAPLL_CONFIG	0xc8	8	R/W
BU_BM_FRONT_BYPASS	0xca	1	R/W

Table C.11.1 Top-Level Registers A Top Level Address Map (contd)

C.12.1 Address Generator Keyhole Space

Notes on address generator keyhole table:

1) All registers in the address generator keyhole take up 4 bytes of address space regardless of their width. The missing addresses (0x00, 0x04 etc.) will always read back zero.

2) The access bit of the relevant block (dispaddr or waddrgen) must be set before accessing this keyhole.

Table C.11.2 Top-Lev 1 RegistersA
Address Generator Keyh le

Keyhole Register Name	Keyhole Address	Bits	Comments
BU_DISPADDR_COMP0_OFFSET_MSB	0x25	2	Must be
BU_DISPADDR_COMP0_OFFSET_MID	0x25	8	Loaded
BU_DISPADDR_COMP0_OFFSET_LSB	0x27	8	
BU_DISPADDR_COMP1_OFFSET_MSB	0x29	2	Must be
BU_DISPADDR_COMP1_OFFSET_MID	0x2a	8	Loaded
BU_DISPADDR_COMP1_OFFSET_LSB	0x2b	8	
BU_DISPADDR_COMP2_OFFSET_MSB	0x2d	2	Must be
BU_DISPADDR_COMP2_OFFSET_MID	0x2e	8	Loaded
BU_DISPADDR_COMP2_OFFSET_LSB	0x2f	8	
BU_DISPADDR_COMP0_VBS_MSB	0x31	2	Must be
BU_DISPADDR_COMP0_VBS_MID	0x32	8	Loaded
BU_DISPADDR_COMP0_VBS_LSB	0x33	8	
BU_DISPADDR_COMP1_VBS_MSB	0x35	2	Must be
BU_DISPADDR_COMP1_VBS_MID	0x36	8	Loaded
BU_DISPADDR_COMP1_VBS_LSB	0x37	8	
BU_DISPADDR_COMP2_VBS_MSB	0x39	2	Must be
BU_DISPADDR_COMP2_VBS_MID	0x3a	8	Loaded
BU_DISPADDR_COMP2_VBS_LSB	0x3b	8	
BU_ADDR_COMP0_HBS_MSB	0x3d	2	Must be
BU_ADDR_COMP0_HBS_MID	0x3e	8	Loaded
BU_ADDR_COMP0_HBS_LSB	0x3f	8	
BU_ADDR_COMP1_HBS_MSB	0x41	2	Must be
BU_ADDR_COMP1_HBS_MID	0x42	8	Loaded
BU_ADDR_COMP1_HBS_LSB	0x43	8	
BU_ADDR_COMP2_HBS_MSB	0x45	2	Must be
BU_ADDR_COMP2_HBS_MID	0x46	8	Loaded
BU_ADDR_COMP2_HBS_LSB	0x47	8	
BU_DISPADDR_COMP0_HBS_MSB	0x49	2	Must be
BU_DISPADDR_COMP0_HBS_MID	0x4a	8	Loaded
BU_DISPADDR_COMP0_HBS_LSB	0x4b	8	
BU_DISPADDR_COMP1_HBS_MSB	0x4d	2	Must be
BU_DISPADDR_COMP1_HBS_MID	0x4e	8	Loaded
BU_DISPADDR_COMP1_HBS_LSB	0x4f	8	

Tabl C.11.2 Top-L vel RegistersA Address Generator Keyhol

Keyhole Register Name	Keyhole Address	Bits	Comments
BU_DISPADDR_BLOCKS_LEFT1_MSB	0x7d	2	Test only
BU_DISPADDR_BLOCKS_LEFT1_MID	0x7e	8	
BU_DISPADDR_BLOCKS_LEFT1_LSB	0x7f	8	
BU_DISPADDR_BLOCKS_LEFT2_MSB	0x81	2	Test only
BU_DISPADDR_BLOCKS_LEFT2_MID	0x82	8	
BU_DISPADDR_BLOCKS_LEFT2_LSB	0x83	8	
BU_WADDR_BUFFER0_BASE_MSB	0x85	2	Must be
BU_WADDR_BUFFER0_BASE_MID	0x86	8	Loaded
BU_WADDR_BUFFER0_BASE_LSB	0x87	8	
BU_WADDR_BUFFER1_BASE_MSB	0x89	2	Must be
BU_WADDR_BUFFER1_BASE_MID	0x8a	8	Loaded
BU_WADDR_BUFFER1_BASE_LSB	0x8b	8	
BU_WADDR_BUFFER2_BASE_MSB	0x8d	2	Must be
BU_WADDR_BUFFER2_BASE_MID	0x8e	8	Loaded
BU_WADDR_BUFFER2_BASE_LSB	0x8f	8	
BU_WADDR_COMP0_HMBADDR_MSB	0x91	2	Test only
BU_WADDR_COMP0_HMBADDR_MID	0x92	8	
BU_WADDR_COMP0_HMBADDR_LSB	0x93	8	
BU_WADDR_COMP1_HMBADDR_MSB	0x95	2	Test only
BU_WADDR_COMP1_HMBADDR_MID	0x96	8	
BU_WADDR_COMP1_HMBADDR_LSB	0x97	8	
BU_WADDR_COMP2_HMBADDR_MSB	0x99	2	Test only
BU_WADDR_COMP2_HMBADDR_MID	0x9a	8	
BU_WADDR_COMP2_HMBADDR_LSB	0x9b	8	
BU_WADDR_COMP0_VMBADDR_MSB	0x9d	2	Test only
BU_WADDR_COMP0_VMBADDR_MID	0x9e	8	
BU_WADDR_COMP0_VMBADDR_LSB	0x9f	8	
BU_WADDR_COMP1_VMBADDR_MSB	0xa1	2	Test only
BU_WADDR_COMP1_VMBADDR_MID	0xa2	8	
BU_WADDR_COMP1_VMBADDR_LSB	0xa3	8	
BU_WADDR_COMP2_VMBADDR_MSB	0xa5	2	Test only
BU_WADDR_COMP2_VMBADDR_MID	0xa6	8	
BU_WADDR_COMP2_VMBADDR_LSB	0xa7	8	

Table C.11.2 Top-Level Regist rsA Addr ss G nerator K yhole

Keyhole Register Name	Keyhole Address	Bits	Comments
BU_WADDR_COMP0_LAST_MB_IN_ROW_MSB	0xd5	2	Must be
BU_WADDR_COMP0_LAST_MB_IN_ROW_MID	0xd6	8	Loaded
BU_WADDR_COMP0_LAST_MB_IN_ROW_LSB	0xd7	8	
BU_WADDR_COMP1_LAST_MB_IN_ROW_MSB	0xd9	2	Must be
BU_WADDR_COMP1_LAST_MB_IN_ROW_MID	0xda	8	Loaded
BU_WADDR_COMP1_LAST_MB_IN_ROW_LSB	0xdb	8	
BU_WADDR_COMP2_LAST_MB_IN_ROW_MSB	0xdd	2	Must be
BU_WADDR_COMP2_LAST_MB_IN_ROW_MID	0xde	8	Loaded
BU_WADDR_COMP2_LAST_MB_IN_ROW_LSB	0xdf	8	
BU_WADDR_COMP0_LAST_MB_IN_HALF_ROW_MSB	0xe1	2	Must be
BU_WADDR_COMP0_LAST_MB_IN_HALF_ROW_MID	0xe2	8	Loaded
BU_WADDR_COMP0_LAST_MB_IN_HALF_ROW_LSB	0xe3	8	
BU_WADDR_COMP1_LAST_MB_IN_HALF_ROW_MSB	0xe5	2	Must be
BU_WADDR_COMP1_LAST_MB_IN_HALF_ROW_MID	0xe6	8	Loaded
BU_WADDR_COMP1_LAST_MB_IN_HALF_ROW_LSB	0xe7	8	
BU_WADDR_COMP2_LAST_MB_IN_HALF_ROW_MSB	0xe9	2	Must be
BU_WADDR_COMP2_LAST_MB_IN_HALF_ROW_MID	0xea	8	Loaded
BU_WADDR_COMP2_LAST_MB_IN_HALF_ROW_LSB	0xeb	8	
BU_WADDR_COMP0_LAST_ROW_IN_MB_MSB	0xed	2	Must be
BU_WADDR_COMP0_LAST_ROW_IN_MB_MID	0xee	8	Loaded
BU_WADDR_COMP0_LAST_ROW_IN_MB_LSB	0xef	8	
BU_WADDR_COMP1_LAST_ROW_IN_MB_MSB	0xf1	2	Must be
BU_WADDR_COMP1_LAST_ROW_IN_MB_MID	0xf2	8	Loaded
BU_WADDR_COMP1_LAST_ROW_IN_MB_LSB	0xf3	8	
BU_WADDR_COMP2_LAST_ROW_IN_MB_MSB	0xf5	2	Must be
BU_WADDR_COMP2_LAST_ROW_IN_MB_MID	0xf6	8	Loaded
BU_WADDR_COMP2_LAST_ROW_IN_MB_LSB	0xf7	8	
BU_WADDR_COMP0_BLOCKS_PER_MB_ROW_MSB	0xf9	2	Must be
BU_WADDR_COMP0_BLOCKS_PER_MB_ROW_MID	0xfa	8	Loaded
BU_WADDR_COMP0_BLOCKS_PER_MB_ROW_LSB	0xfb	8	
BU_WADDR_COMP1_BLOCKS_PER_MB_ROW_MSB	0xfd	2	Must be
BU_WADDR_COMP1_BLOCKS_PER_MB_ROW_MID	0xfe	8	Loaded
BU_WADDR_COMP1_BLOCKS_PER_MB_ROW_LSB	0xff	8	

Table C.11.3 H-Upsamplers and Cspace Keyhol Address Map

Keyhole Register Name	Keyhole Address	Bits	Comment
BU_UH0_A00_1	0x0	5	R/W- Coeff 0,0
BU_UH0_A00_0	0x1	8	
BU_UH0_A01_1	0x2	5	R/W- Coeff 0,1
BU_UH0_A01_0	0x3	8	
BU_UH0_A02_1	0x4	5	R/W- Coeff 0,2
BU_UH0_A02_0	0x5	8	
BU_UH0_A03_1	0x6	5	R/W- Coeff 0,0
BU_UH0_A03_0	0x7	8	
BU_UH0_A10_1	0x8	5	R/W- Coeff 1,0
BU_UH0_A10_0	0x9	8	
BU_UH0_A11_1	0xa	5	R/W- Coeff 1,1
BU_UH0_A11_0	0xb	8	
BU_UH0_A12_1	0xc	5	R/W- Coeff 1,2
BU_UH0_A12_0	0xd	8	
BU_UH0_A13_1	0xe	5	R/W- Coeff 1,3
BU_UH0_A13_0	0xf	8	
BU_UH0_A20_1	0x10	5	R/W- Coeff 2,0
BU_UH0_A20_0	0x11	8	
BU_UH0_A21_1	0x12	5	R/W- Coeff 2,1
BU_UH0_A21_0	0x13	8	
BU_UH0_A22_1	0x14	5	R/W- Coeff 2,2
BU_UH0_A22_0	0x15	8	
BU_UH0_A23_1	0x16	5	R/W- Coeff 2,3
BU_UH0_A23_0	0x17	8	
BU_UH0_MODE	0x18	2	R/W
BU_UH1_A00_1	0x20	5	R/W- Coeff 0,0
BU_UH1_A00_0	0x21	8	
BU_UH1_A01_1	0x22	5	R/W- Coeff 0,1
BU_UH1_A01_0	0x23	8	
BU_UH1_A02_1	0x24	5	R/W- Coeff 0,2
BU_UH1_A02_0	0x25	8	
BU_UH1_A03_1	0x26	5	R/W- Coeff 0,0
BU_UH1_A03_0	0x27	8	

Table C.11.3 H-Upsamplers and Cspace Keyhole Address Map

Keyhole Register Name	Keyhole Address	Bits	Comment
BU_UH2_A20_1	0x50	5	R/W- Coeff 2.0
BU_UH2_A20_0	0x51	8	
BU_UH2_A21_1	0x52	5	R/W- Coeff 2.1
BU_UH2_A21_0	0x53	8	
BU_UH2_A22_1	0x54	5	R/W- Coeff 2.2
BU_UH2_A22_0	0x55	8	
BU_UH2_A23_1	0x56	5	R/W- Coeff 2.3
BU_UH2_A23_0	0x57	8	
BU_UH2_MODE	0x58	2	R/W
BU_CS_A00_1	0x60	5	R/W
BU_CS_A00_0	0x61	8	
BU_CS_A10_1	0x62	5	R/W
BU_CS_A10_0	0x63	8	
BU_CS_A20_1	0x64	5	R/W
BU_CS_A20_0	0x65	8	
BU_CS_B0_1	0x66	6	R/W
BU_CS_B0_0	0x67	8	
BU_CS_A01_1	0x68	5	R/W
BU_CS_A01_0	0x69	8	
BU_CS_A11_1	0x6a	5	R/W
BU_CS_A11_0	0x6b	8	
BU_CS_A21_1	0x6c	5	R/W
BU_CS_A21_0	0x6d	8	
BU_CS_B1_1	0x6e	6	R/W
BU_CS_B1_0	0x6f	8	
BU_CS_A02_1	0x70	5	R/W
BU_CS_A02_0	0x71	8	
BU_CS_A12_1	0x72	5	R/W
BU_CS_A12_0	0x73	8	
BU_CS_A22_1	0x74	5	R/W
BU_CS_A22_0	0x75	8	
BU_CS_B2_1	0x76	6	R/W
BU_CS_B2_0	0x77	8	

```
if (hmbs_event)
 load(mbs_wide);
else if (vmbs_event)
 load(mbs_high);
else if (def_samp0_event)
{
 load (maxhb[0]);
 load (maxvb[0]);
}
else if (def_samp1_event)
{
 load (maxhb[1]);
 load (maxvb[1]);
}
else if (def_samp2_event)
{
 load (maxhb[2]);
 load (maxvb[2]);
}
```

In addition, the following calculations are necessary to retain consistent picture size parameters:

```
if (hmbs_event||vmbs_event||
    def_samp0_event||def_samp1_event||def_samp2_event)
{
 for (i=0; i<max_component; i++)
 {
  hbs[i] = addr_hbs[i] = (maxhb[i]+1) * mbs_wide;
  half_width_in_blocks[i] = ((maxhb[i]+1) * mbs_wide)/2;
  last_mb_in_row[i] = hbs[i] - (maxhb[i]+1);
  last_mb_in_half_row[i] = half_width_in_blocks[i] -
(maxhb[i]+1);
  last_row_in_mb[i] = hbs[i] * maxvb[i];
  blocks_per_mb_row[i] = last_row_in_mb[i] + hbs[i];
  last_mb_row[i] = blocks_per_mb_row[i] * (mbs_high-1);
 }
}
```

BU_WADDR_COMP1_LAST_MB_IN_ROW	= 0x15
BU_WADDR_COMP2_LAST_MB_IN_ROW	= 0x15
BU_WADDR_COMP0_LAST_MB_IN_HALF_ROW	= 0x14
BU_WADDR_COMP1_LAST_MB_IN_HALF_ROW	= 0x0A
BU_WADDR_COMP2_LAST_MB_IN_HALF_ROW	= 0x0A
BU_WADDR_COMP0_LAST_ROW_IN_MB	= 0x2C
BU_WADDR_COMP1_LAST_ROW_IN_MB	= 0x0
BU_WADDR_COMP2_LAST_ROW_IN_MB	= 0x0
BU_WADDR_COMP0_BLOCKS_PER_MB_ROW	= 0x58
BU_WADDR_COMP1_BLOCKS_PER_MB_ROW	= 0x16
BU_WADDR_COMP2_BLOCKS_PER_MB_ROW	= 0x16
BU_WADDR_COMP0_LAST_MB_ROW	= 0x5D8
BU_WADDR_COMP1_LAST_MB_ROW	= 0x176
BU_WADDR_COMP2_LAST_MB_ROW	= 0x176

Note that if these values are to be written explicitly at setup, account must be taken of the multi-byte nature of most of the locations.

Note that additional Figures, which are self explanatory to those of ordinary skill in the art, are included with this application for providing further insight into the detailed structure and operation of the environment in which the present invention is intended to function.

Accordingly, it is not intended that th invention be limited, except as by the appended claims.

7. A pipeline machine as recited in any of claims 1-4,
wherein said tokens ar alter d by interfacing with said stages.

8. A pipeline machine as recited in any of claims 1-4,
wherein said tokens interact with all of said stages.

9. A pipeline machine as recited in any of claims 1-4,
wherein said tokens interact with some but less than all of said processing stages.

10. A pipeline machine as recited in any of claims 1-4,
wherein said tokens interact with only predetermined ones of said processing stages.

11. A pipeline machine as recited in any of claims 1-4,
wherein said tokens interact with adjacent processing stages.

12. A pipeline machine as recited in any of claims 1-4,
wherein said tokens interact with non-adjacent processing stages.

13. A pipeline machine as recited in any of claims 1-4,
wherein said tokens reconfigure said processing stages.

21. A pipelin machin as recited in claim 20, wherein said extension bit identifi s the last word in said tok n.

22. A pipeline machine as recited in claim 18, wherein said address fields are of variable length.

23. A pipeline machine as recited in claim 17, wherein said address fields are Huffman coded.

24. A pipeline machine as recited in any of claims 1-4,
wherein said tokens are generated by a processing stage.

25. A pipeline machine as recited in any of claims 1-4,
wherein said tokens include data for transfer to said processing stages in said pipeline.

26. A pipeline machine as recited in any of claims 1-4,
wherein said tokens are devoid of data.

27. A pipeline machine as recited in any of claims 1-4,
wherein some of said tokens are identified as DATA tokens and provide data to said processing stages in said pipeline.

28. A pipeline machine as recited in any of claims 1-4,
wherein some of said tokens are identified as control tokens and only condition said processing stages in said pipeline.

36. A pipelin machin as recited in any of claims 1-4,

wherein said tokens are capable of facilitating a plurality of functions within any processing stage in said pipeline.

37. A pipeline machine as recited in any of claims 1-4,

wherein said tokens are hardware based.

38. A pipeline machine as recited in any of claims 1-4,

wherein said tokens are software based.

39. A pipeline machine as recited in any of claims 1-4,

wherein said tokens facilitate more efficient uses of system bandwidth.

40. A pipeline machine as recited in any of claims 1-4,

wherein said tokens provide data and control simultaneously to said processing stages in said pipeline.

41. In a pipeline processing machine for handling plurality of separately encoded bit streams arranged as a single serial bit stream of digital bits and having separately encoded pairs of control codes and corresponding data carried in the serial bit stream and employing a plurality of stages interconnected by a two-wire interface, characterized by: a start code detector responsive to the single serial bit stream for generating control tokens and DATA tokens for application to the two-wire interface; a token decode circuit positioned in certain of said stages f r recognizing certain of said tokens as control tokens pertinent to that stage and for passing unrecognized control tokens along the pipeline;

46. In a pipelin processing machin for handling a plurality of separately encod d bit streams arranged as a single serial bit stream of digital bits and having separately encoded pairs of control codes and corresponding data carried in the serial bit stream and employing a plurality of stages interconnected by a two-wire interface, characterized by: a start code detector responsive to the single serial bit stream for generating control tokens and DATA tokens for application to said two-wire interface; a token decode means positioned in certain of said stages for recognizing certain of said tokens as control tokens pertinent to that stage and for passing unrecognized control tokens along said pipeline; a reconfigurable temporal decoder responsive to a recognized control token for reconfiguring said temporal decoder stage to handle an identified DATA token, data being moved along said two-wire interface within said temporal decoder in 8x8 pel data blocks; and address means for storing and retrieving said blocks along block boundaries.

47. A machine as in claim 46, and further characterized in that said address means store and retrieve blocks of data across block boundaries.

48. A machine as in claim 46, and further characterized in that said address means reorders said blocks as picture data for display.

49. A machine as recited in any one of claims 46-48, and further characterized by: circuit means for either displaying the output of said temporal decoder or writing said output back into a picture memory location.

56. A DATA token as recited in claim 54,
wherein said tok n is position dependent upon said processing stages for its performanc .

57. A DATA token as recited in claim 54,
wherein said token has unlimited word length.

58. A control token as recited in claim 53,
wherein said token has a fixed, invariant word length.

59. A token as recited in either claim 53 or 54,
wherein said token is generated by one of said processing stages.

60. A token as recited in either claim 53 or 54,
wherein said token is created by one of said processing stages.

61. A token as recited in either claim 53 or 54,
wherein said token is converted by one of said processing stages.

62. A token as recited in either claim 53 or 54,
wherein said token is dynamically adaptive.

63. A token as recited in either claim 53 or 54,
wherein said token is altered by interfacing with said stages.

71. A token as recited in either claim 53 or 54,

wherein said token provides a basic building block for the system.

72. A token as recited in either claim 53 or 54,

wherein the interaction of said token with a stage is conditioned by the previous processing history of said stage.

73. A token as recited in either claim 53 or 54,

wherein said token has an address field which characterizes said token.

74. A token as recited in claim 73,

wherein interaction with a selected processing stage is determined by said address field.

75. A token as recited in either claim 53 or 54,

wherein said token includes an extension bit.

76. A token as recited in claim 75,

wherein said extension bit indicates the presence of additional words in said token.

77. A token as recited in either claim 75 or 76,

wherein said extension bit identifies the last word in said token.

78. A token as recited in either claim 73 or 74,

wherein said address field is of variable length.

87. A token as recited in either claim 53 or 54,

wher in said token is capable of successiv alteration by said processing stages.

88. A token as recited in either claim 53 or 54,

wherein the interactive flexibility of said token in cooperation with said processing stages facilitates greater functional diversity of said processing stages for resident structure.

89. A token as recited in either claim 53 or 54,

wherein the flexibility of said token facilitates system expansion and/or alteration.

90. A token as recited in claim 53,

wherein said control token is capable of facilitating a plurality of functions within a processing stage.

91. A token as recited in either claim 53 or 54,

wherein said token is hardware based.

92. A token as recited in either claim 53 or 54,

wherein said token is software based.

93. A token as recited in either claim 53 or 54,

wherein said token facilitates more efficient use of system bandwidth.

Figure (a) to accompany abstract

ABSTRACT

A multi-standard vid o d compr ssion apparatus has a plurality of stages int rconnected by a two-wir int rface arranged as a pipeline processing machine. Control tokens and DATA Tokens pass over the single two-wire interfac for carrying both control and data in token format. A token decode circuit is positioned in certain of the stages for recognizing certain of the tokens as control tokens pertinent to that stage and for passing unrecognized control tokens along the pipeline. Reconfiguration processing circuits are positioned in selected stages and are responsive to a recognized control token for reconfiguring such stage to handle an identified DATA Token. A wide variety of unique supporting subsystem circuitry and processing techniques are disclosed for implementing the system.

FIG. 1

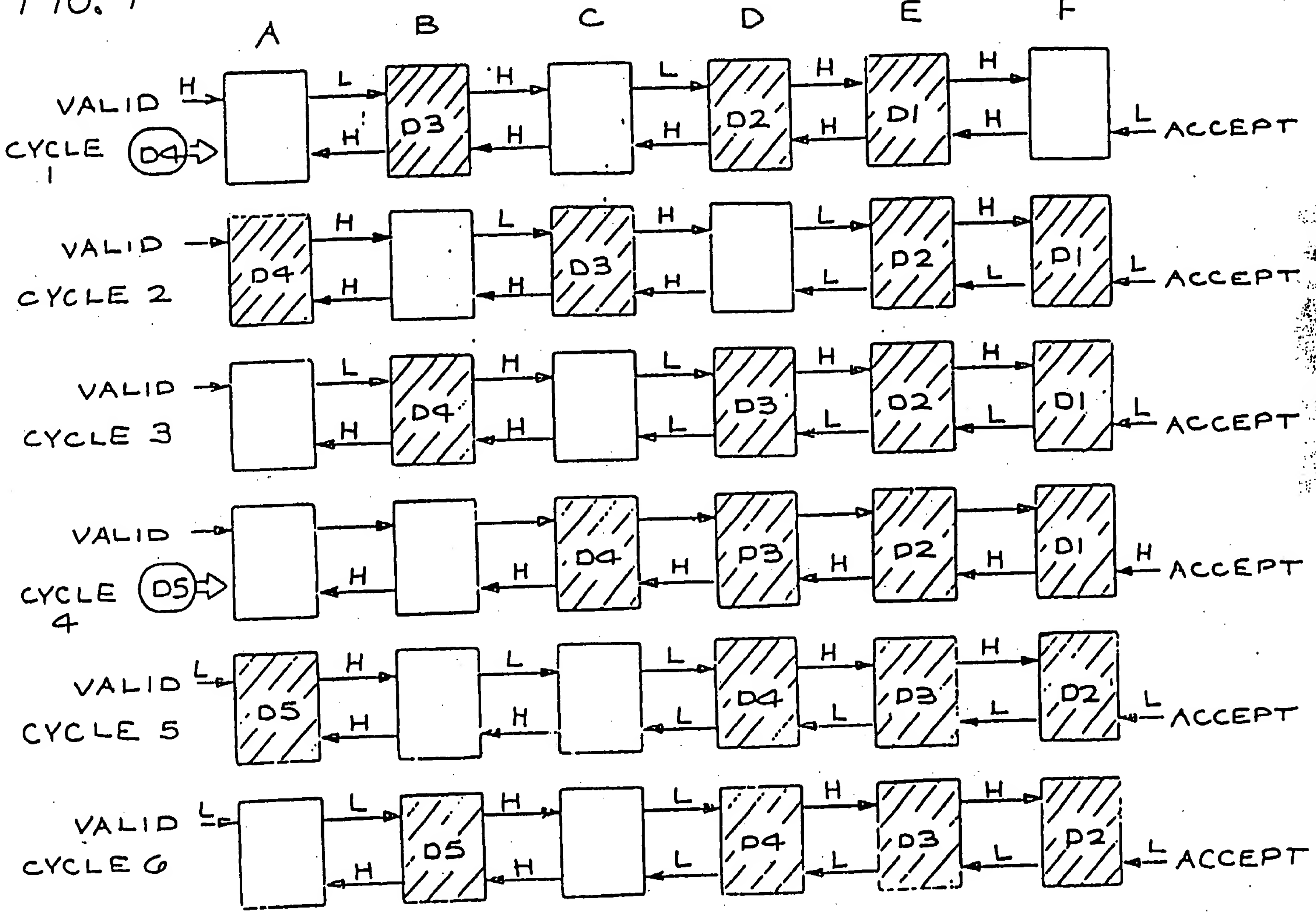
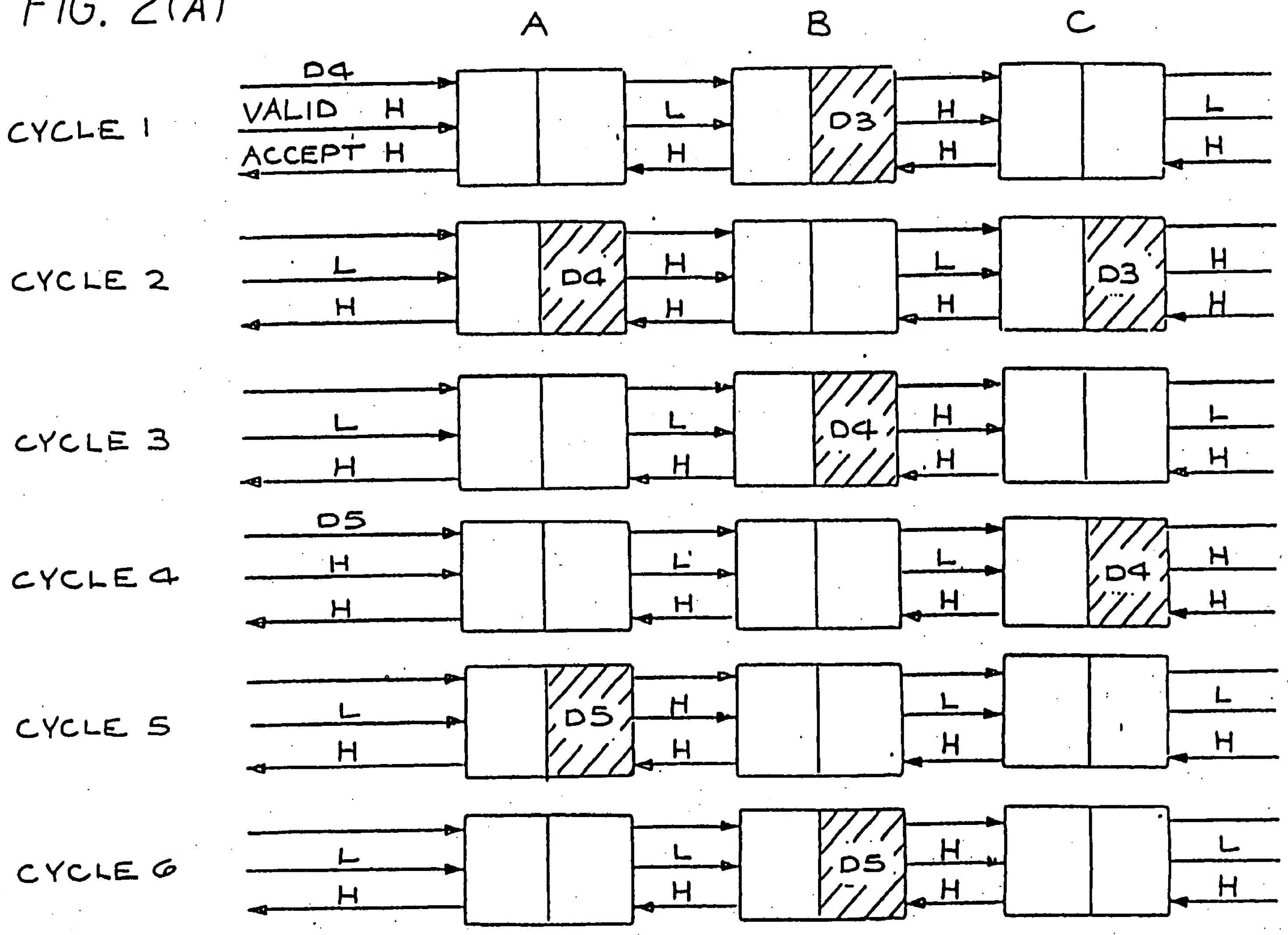

A
B
C
D
E
F
VALID
CYCLE 1
CYCLE 2
CYCLE 3
CYCLE 4
CYCLE 5
CYCLE 6
D1
D2
D3
D4
D5
H
L
ACCEPT

FIG. 2(A)


A
B
C
CYCLE 1
D4
VALID H
ACCEPT H
L
H
D3
H
H
L
H
CYCLE 2
L
H
D4
H
H
L
H
D3
H
H
CYCLE 3
L
H
L
H
D4
H
H
L
H
CYCLE 4
D5
H
H
L
H
L
H
D4
H
H
CYCLE 5
L
H
D5
H
H
L
H
L
H
CYCLE 6
L
H
L
H
D5
H
H
L
H

2

FIG. 2(B)

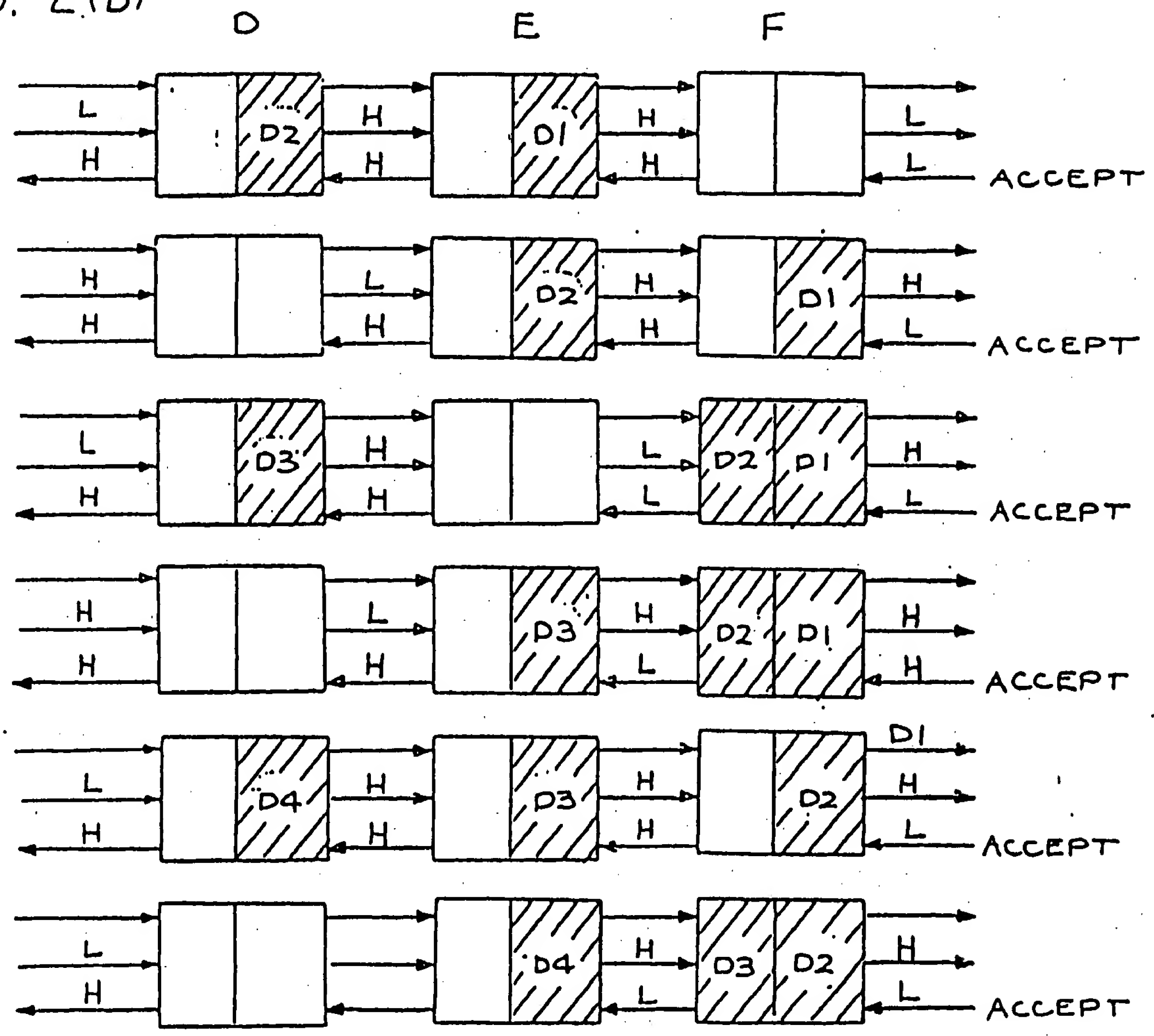

D
E
F
L
H
H
H
H
H
L
L
ACCEPT
D2
D1
D3
D4
D2
D1
D3
D2
D4
D3
D2
D1
D3
D2
D4
D3
D2
D1
ACCEPT
ACCEPT
ACCEPT
ACCEPT
ACCEPT

FIG. 3a(1)

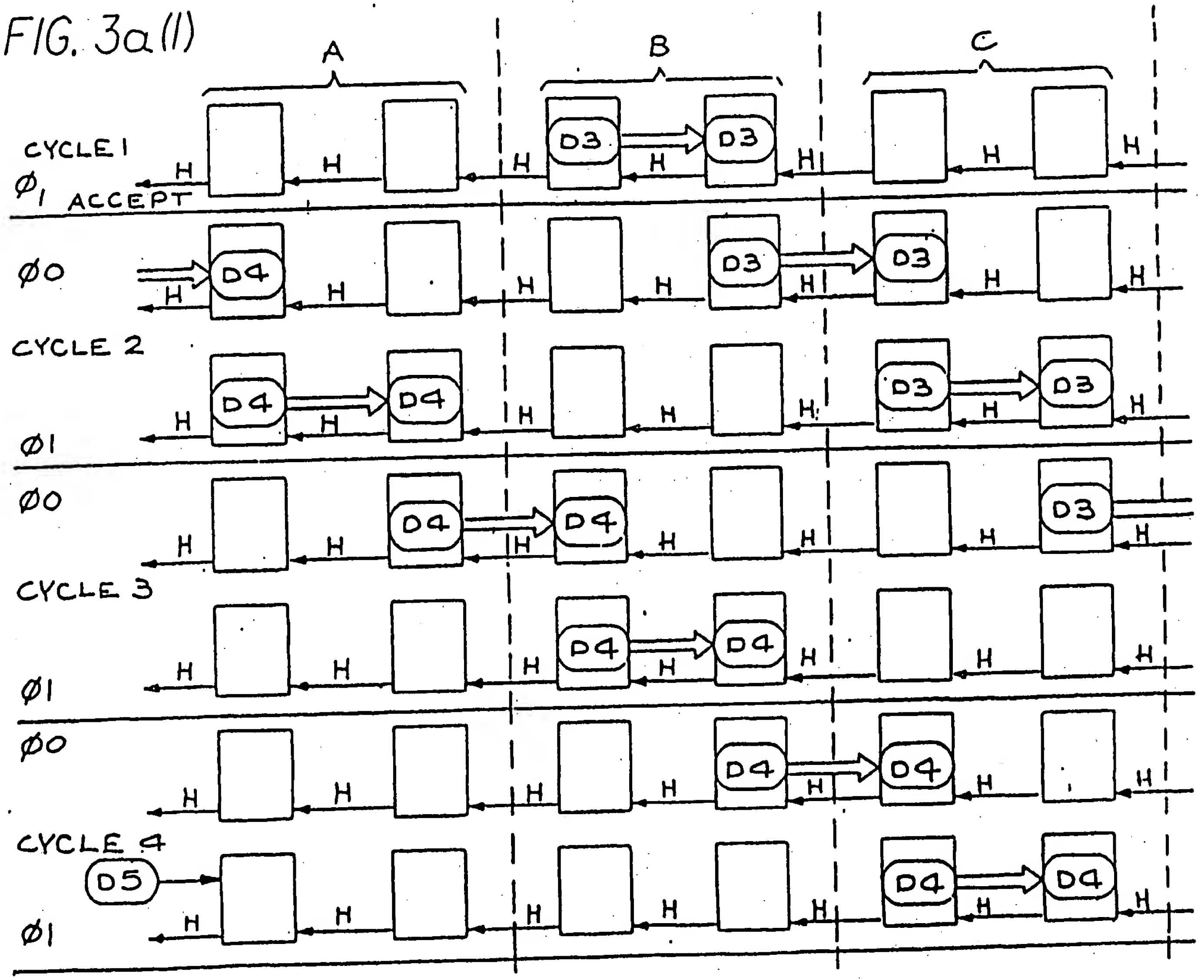

A
B
C
CYCLE 1
ϕ_1 ACCEPT
$\phi 0$
CYCLE 2
$\phi 1$
$\phi 0$
CYCLE 3
$\phi 1$
$\phi 0$
CYCLE 4
$\phi 1$
D3
D4
D5
H

FIG. 3a(2)

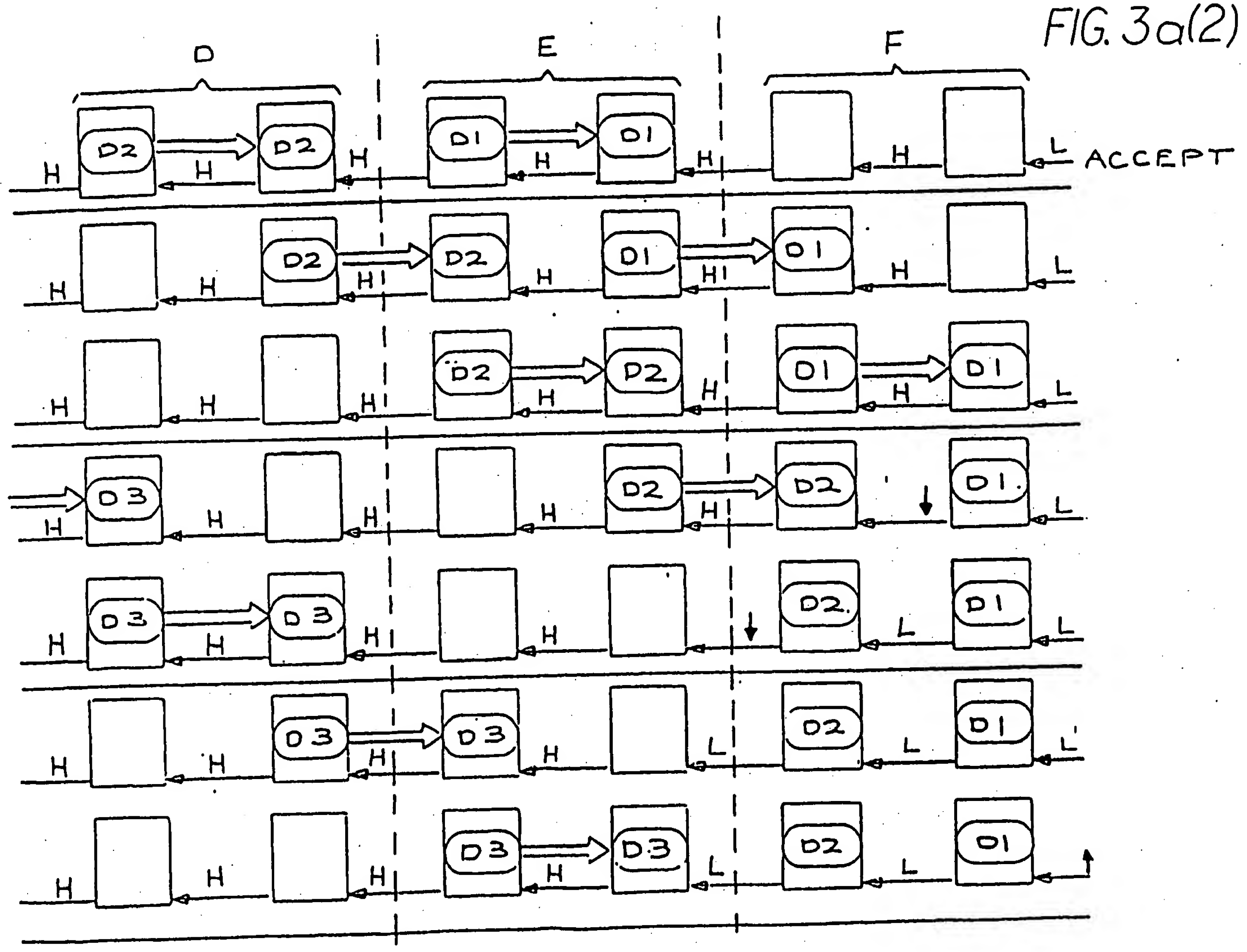
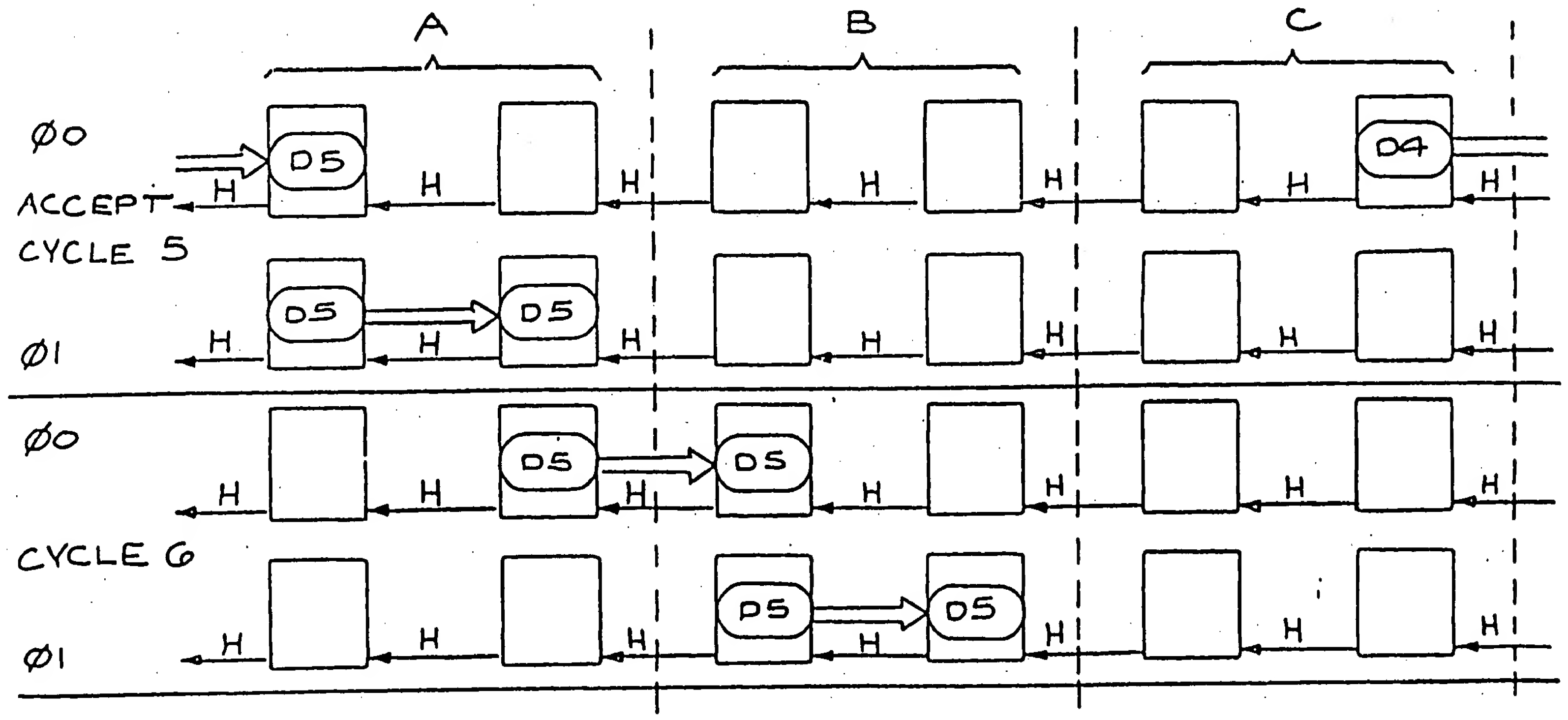
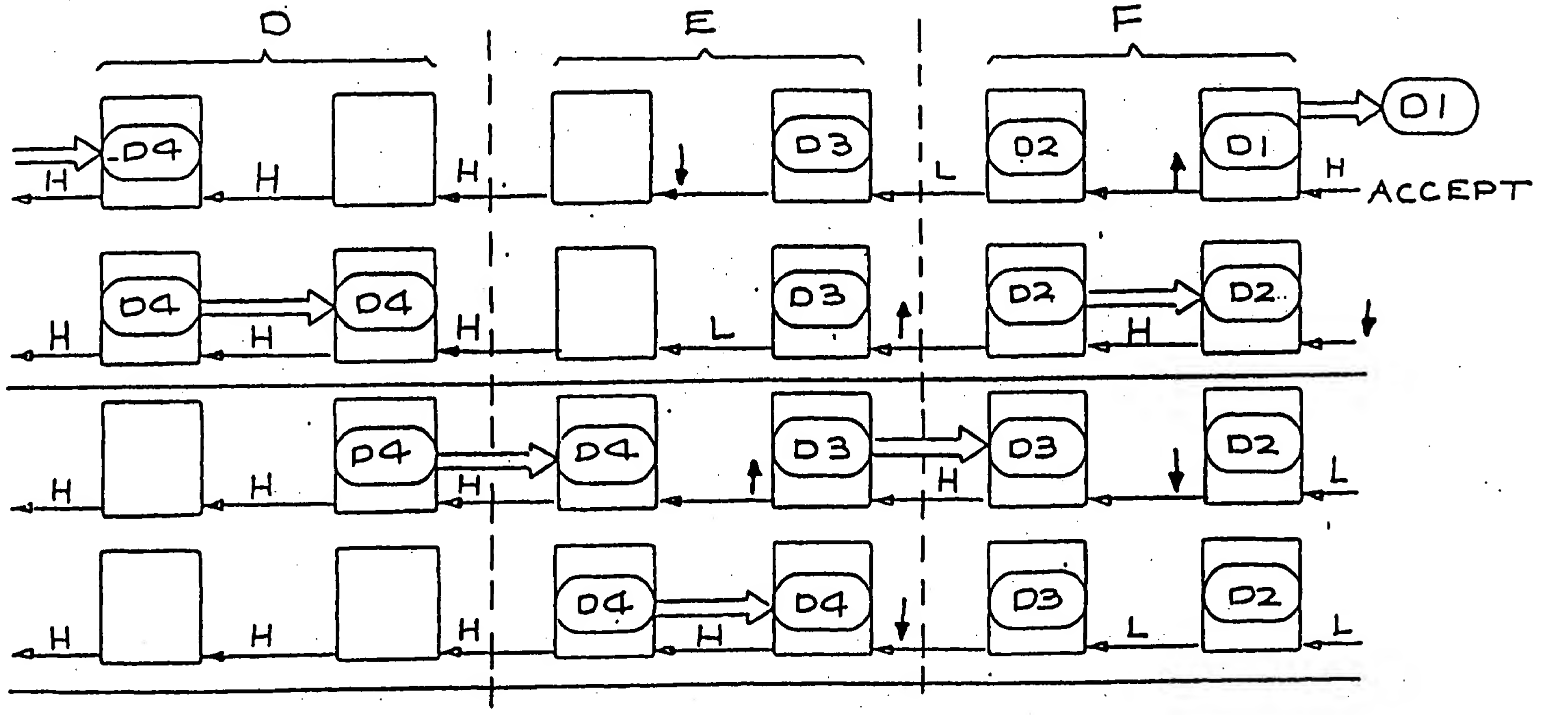


FIG. 3b(1)



6

FIG. 3b(2)



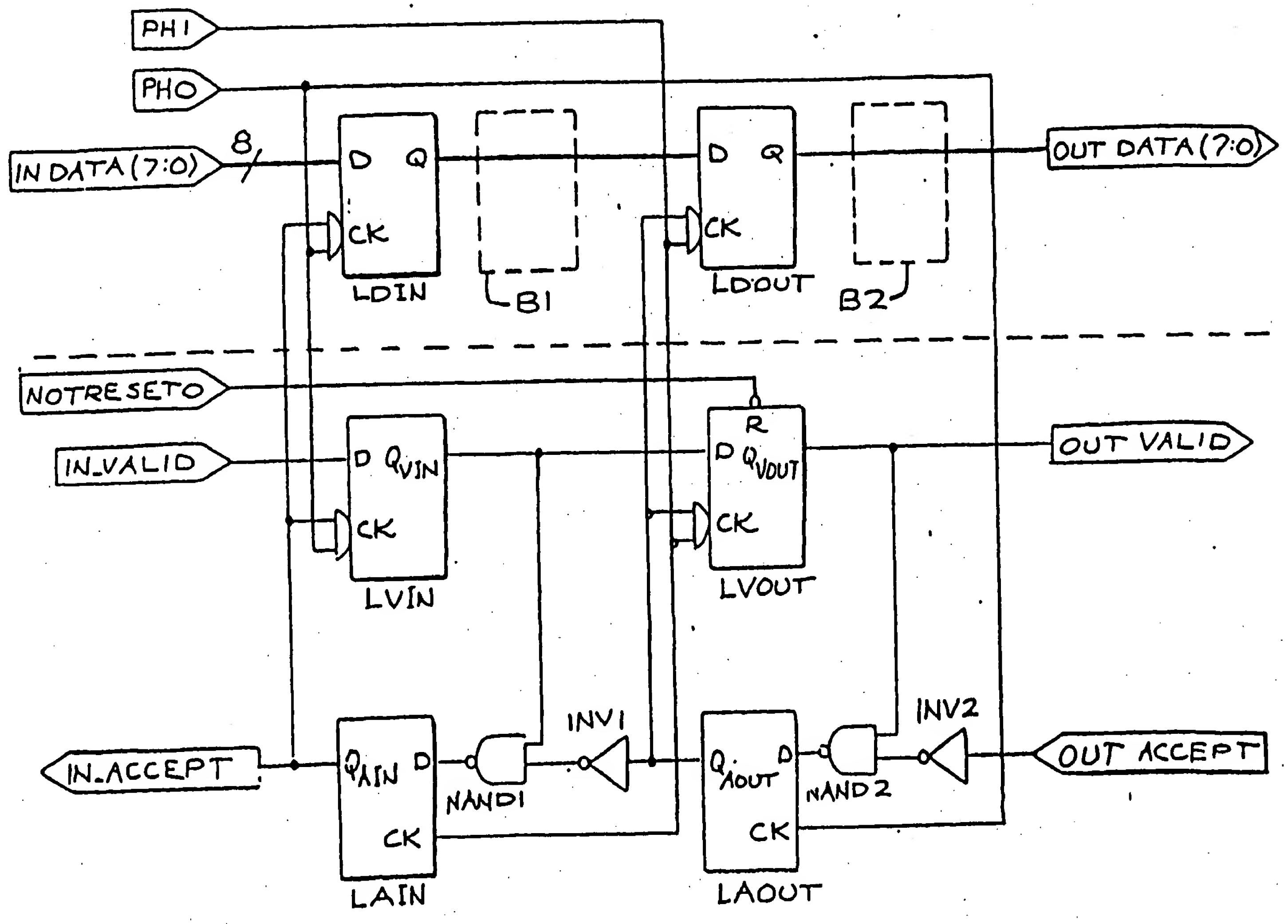

PH1
PH0
IN DATA(7:0)
8
D
Q
CK
LDIN
B1
D
Q
CK
LDOUT
B2
OUT DATA(7:0)
NOTRESET0
IN_VALID
D Q_{VIN}
CK
LVIN
R
D Q_{VOUT}
CK
LVOUT
OUT VALID
IN_ACCEPT
Q_{AIN}
D
CK
LAIN
NAND1
INV1
Q_{AOUT}
D
CK
LAOUT
NAND2
INV2
OUT ACCEPT

FIG. 4

FIG. 5(A)

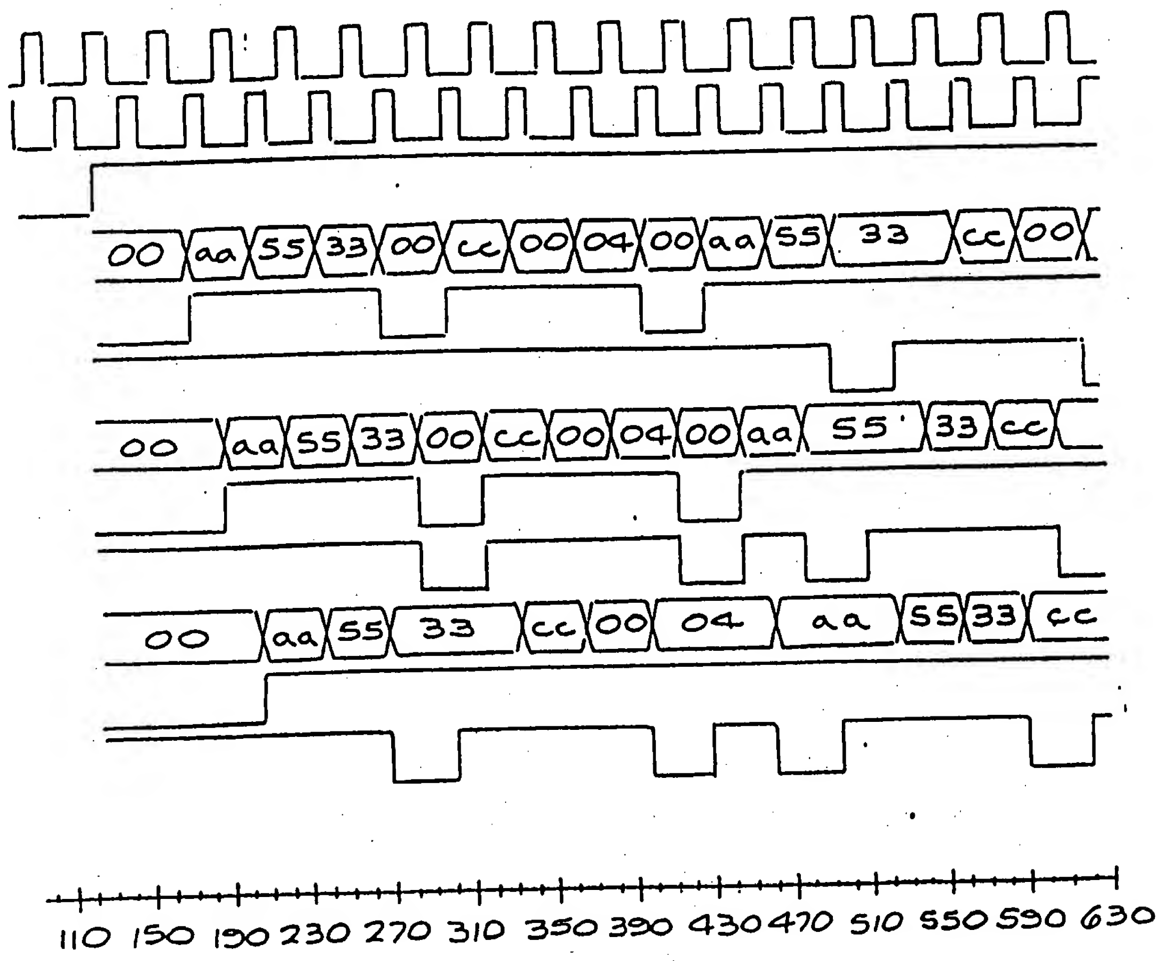
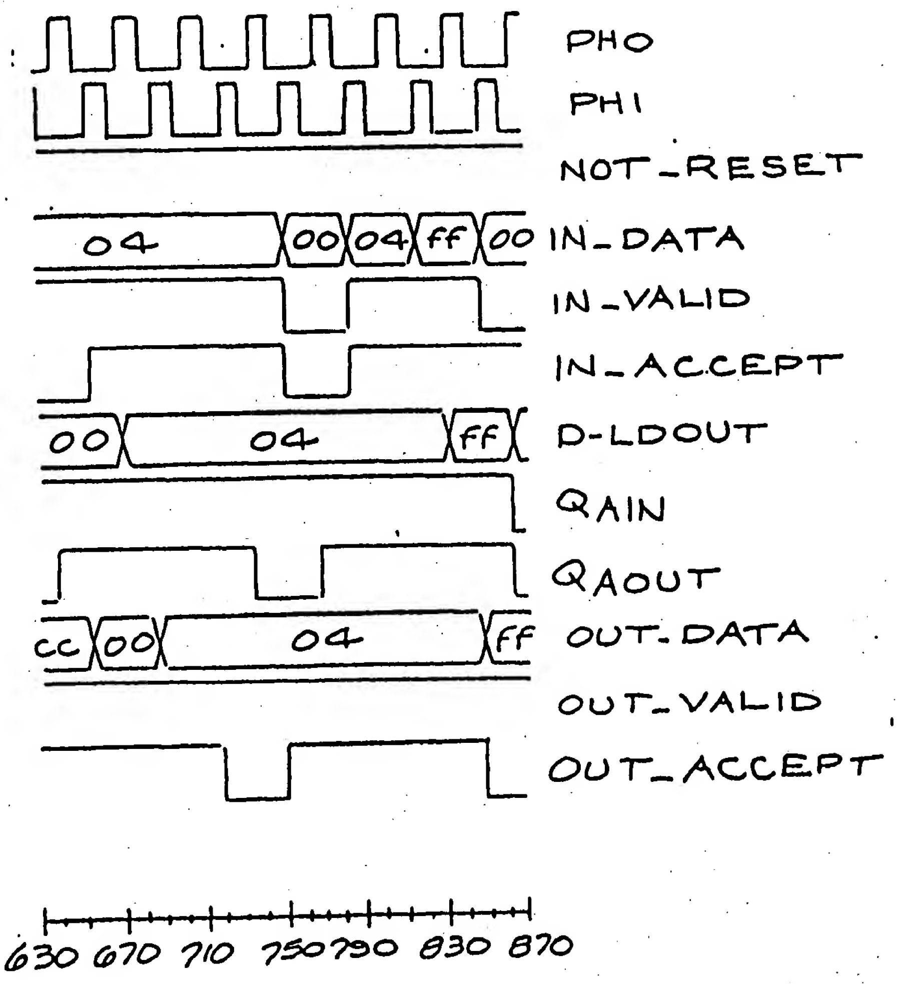

00 aa 55 33 00 cc 00 04 00 aa 55 33 cc 00
00 aa 55 33 00 cc 00 04 00 aa 55 33 cc
00 aa 55 33 cc 00 04 aa 55 33 cc
110 150 190 230 270 310 350 390 430 470 510 550 590 630

FIG. 5(B)



PHO
PHI
NOT_RESET
04
00
04
ff
00
IN_DATA
IN_VALID
IN_ACCEPT
00
04
ff
D-LDOUT
QAIN
QAOUT
cc
00
04
ff
OUT_DATA
OUT_VALID
OUT_ACCEPT
630
670
710
750
790
830
870

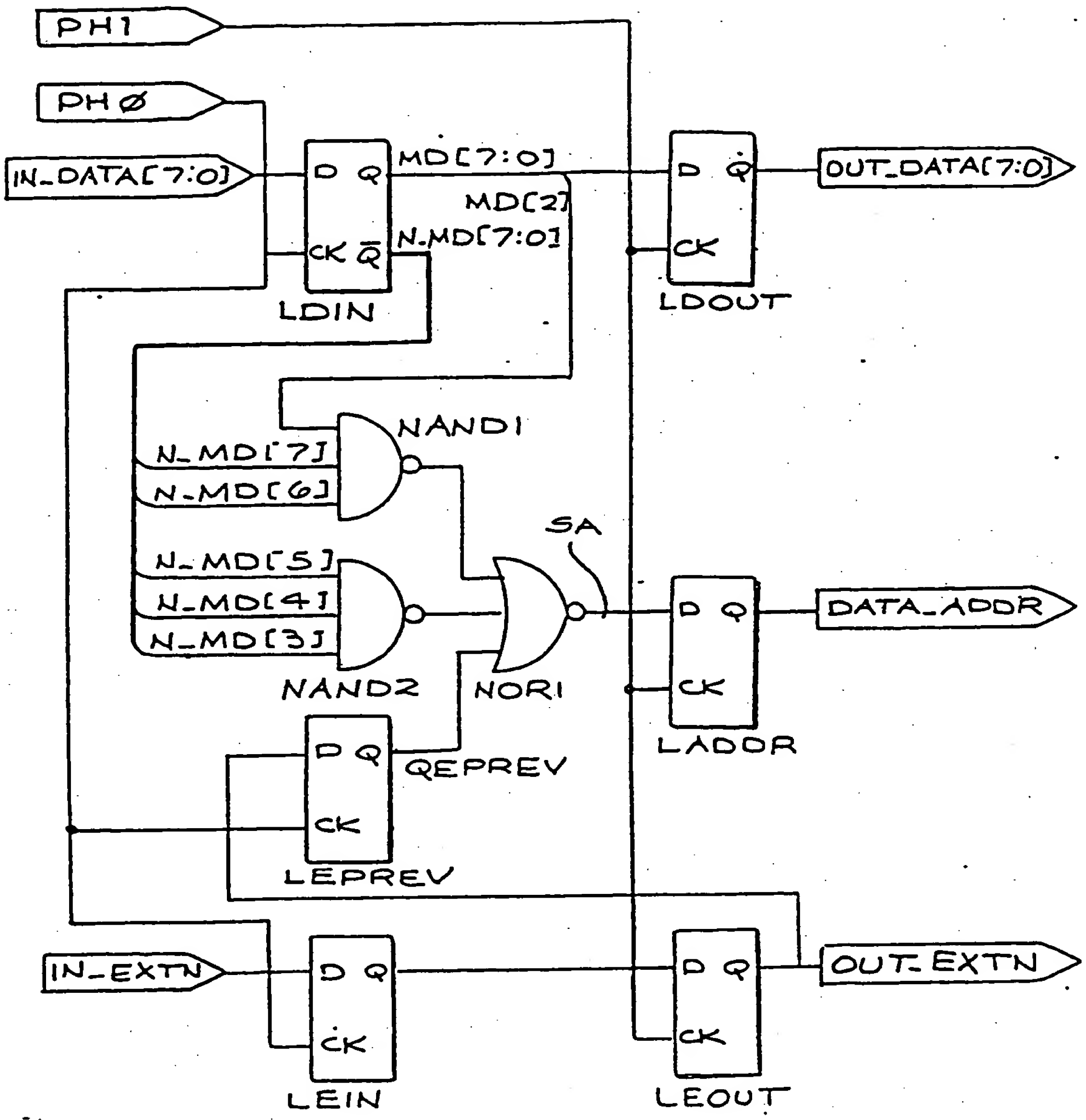

PH1
PHØ
IN_DATA[7:0]
D Q
CK Q̄
LDIN
MD[7:0]
MD[2]
N_MD[7:0]
D Q
CK
LDOUT
OUT_DATA[7:0]
NAND1
N_MD[7]
N_MD[6]
N_MD[5]
N_MD[4]
N_MD[3]
NAND2
SA
NOR1
D Q
CK
LADDR
DATA_ADDR
D Q
CK
QEPREV
LEPREV
IN_EXTN
D Q
CK
LEIN
D Q
CK
LEOUT
OUT_EXTN

FIG. 6

FIG. 7

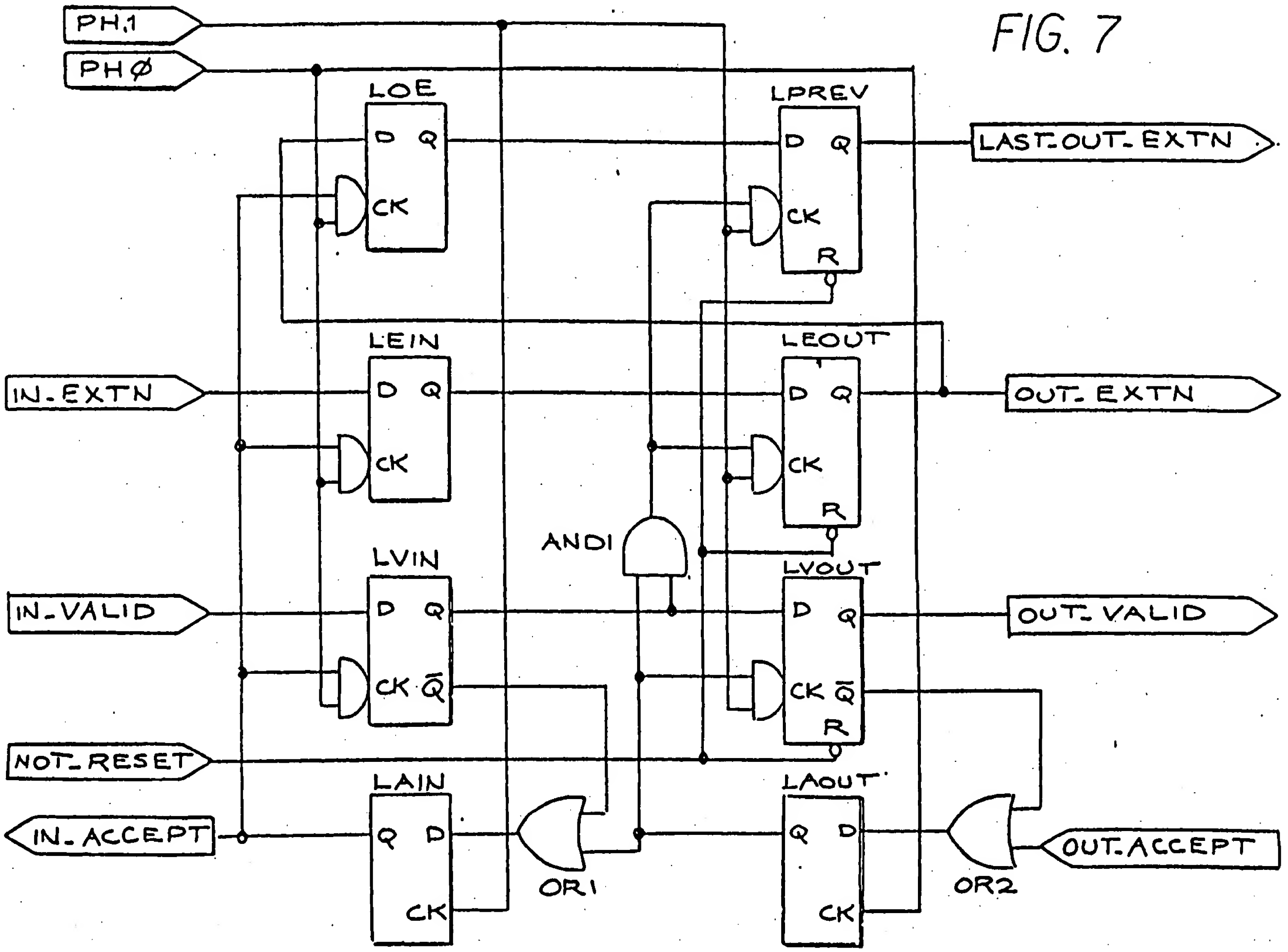
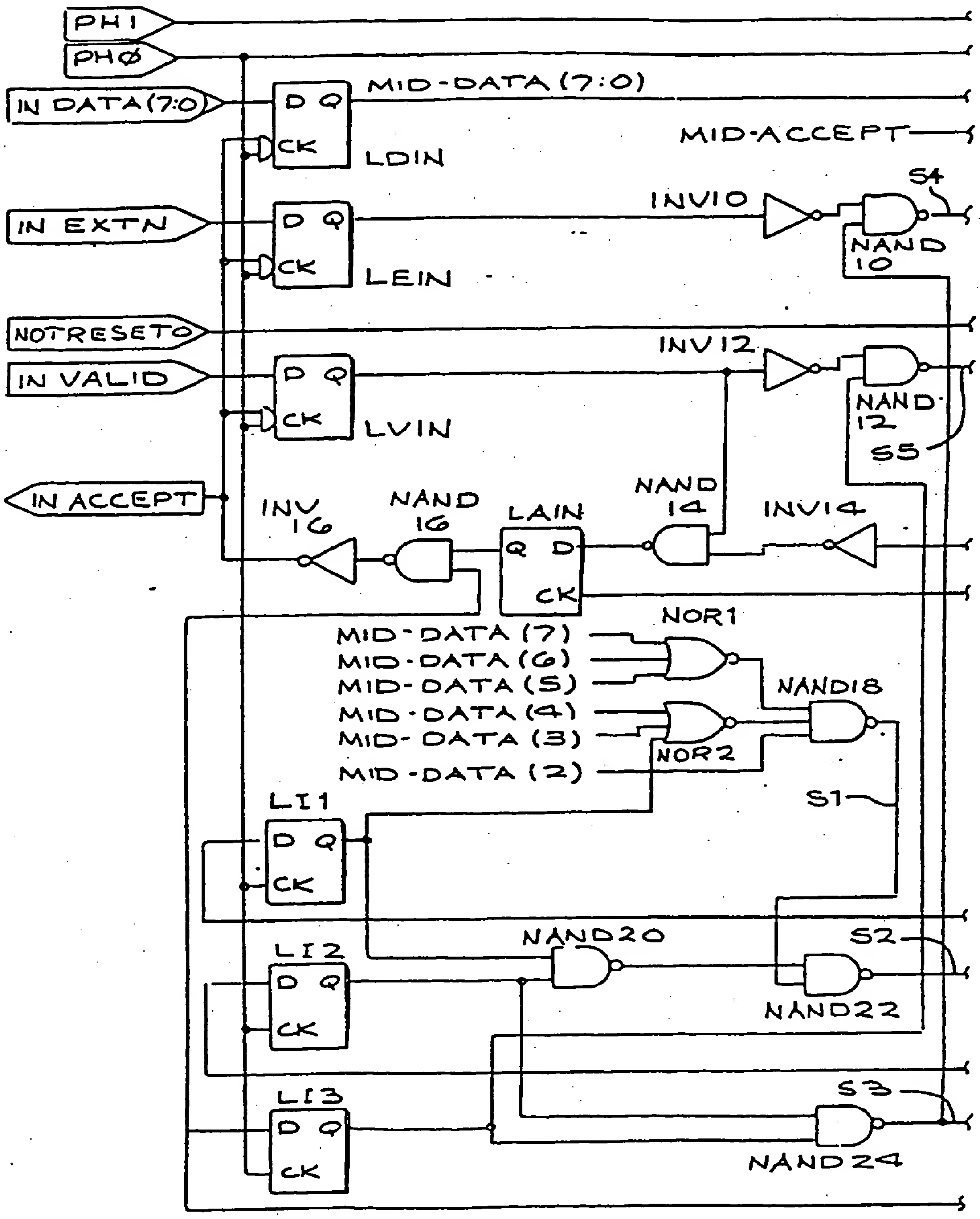

PH.1
PHØ
LOE
LPREV
LAST-OUT-EXTN
LEIN
LEOUT
IN-EXTN
OUT-EXTN
ANDI
LVIN
LVOUT
IN-VALID
OUT-VALID
NOT-RESET
LAIN
LAOUT
IN-ACCEPT
OUT-ACCEPT
OR1
OR2
D
Q
CK
R
$\bar{Q}$

FIG. 8(A)


PH1
PHØ
IN DATA(7:0)
IN EXTN
NOTRESETO
IN VALID
IN ACCEPT
D Q
CK
LDIN
LEIN
LVIN
MID-DATA(7:0)
MID-ACCEPT
INV10
INV12
NAND 10
NAND 12
S4
S5
INV 16
NAND 16
LAIN
NAND 14
INV14
NOR1
NOR2
NAND18
MID-DATA (7)
MID-DATA (6)
MID-DATA (5)
MID-DATA (4)
MID-DATA (3)
MID-DATA (2)
S1
LI1
LI2
LI3
NAND20
NAND22
NAND24
S2
S3

13

FIG. 8(B)

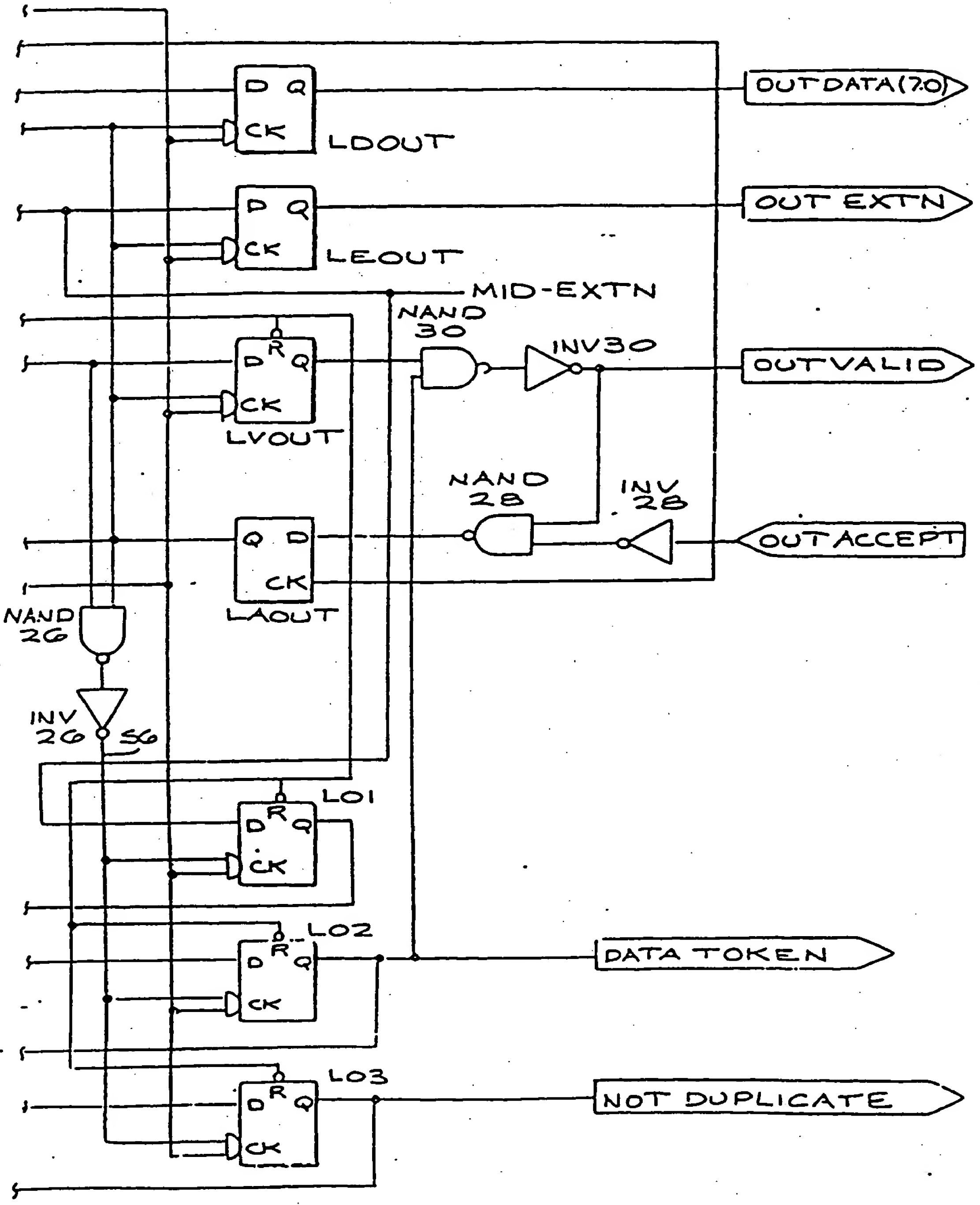
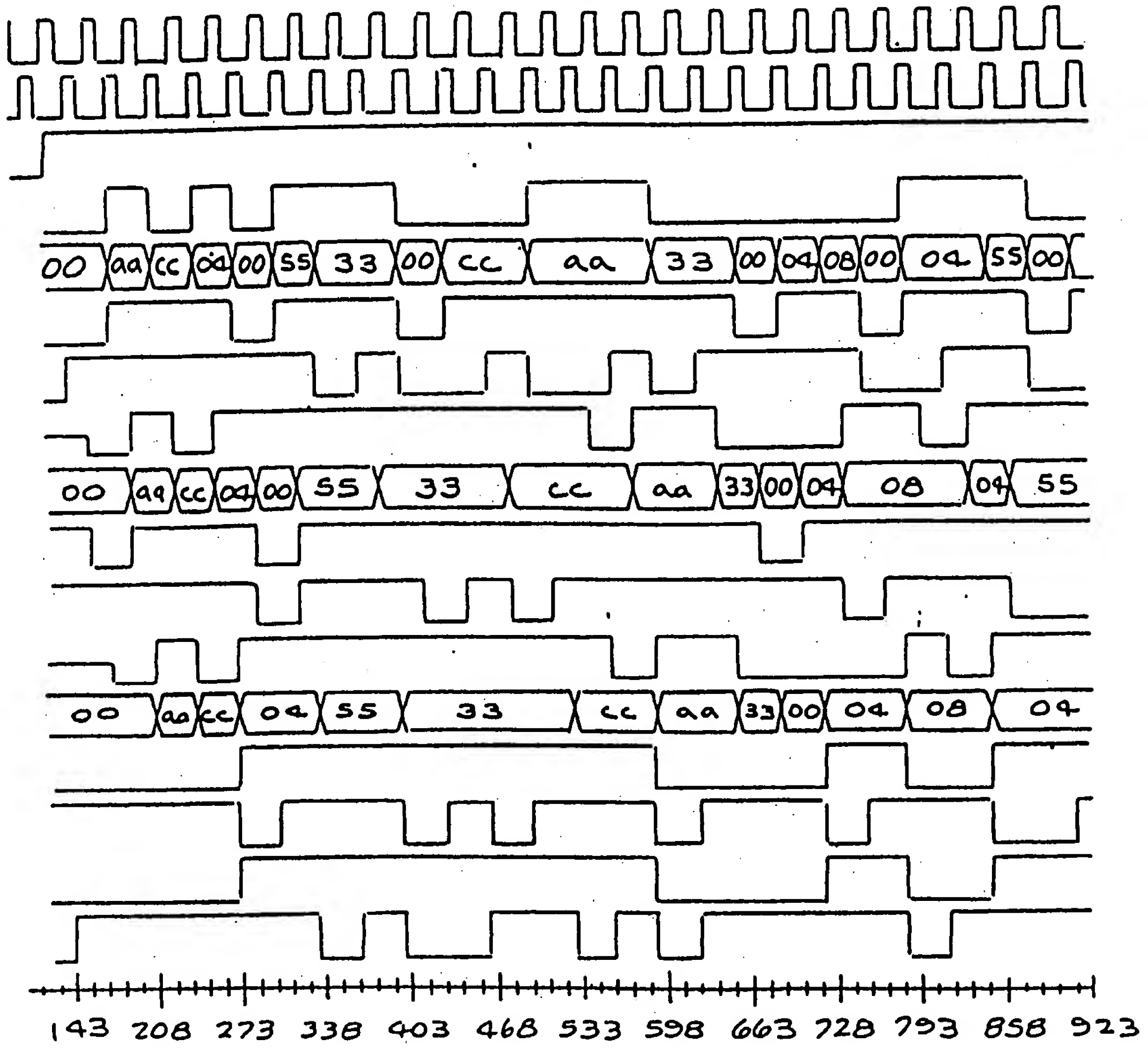
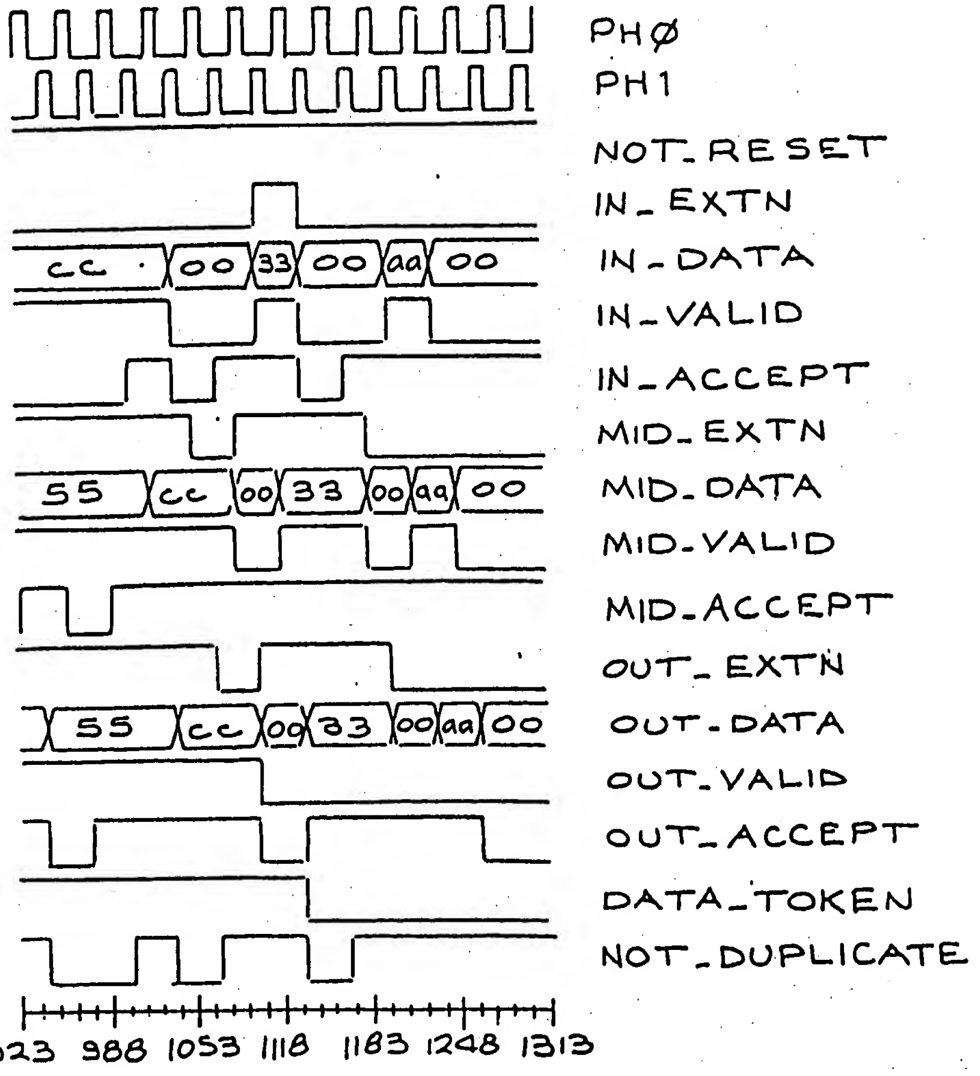

D Q
CK
LDOUT
OUTDATA(7:0)
D Q
CK
LEOUT
OUT EXTN
MID-EXTN
NAND 30
INV30
D R Q
CK
LVOUT
OUTVALID
NAND 28
INV 28
Q D
CK
LAOUT
OUTACCEPT
NAND 26
INV 26
56
LO1
D R Q
CK
LO2
D R Q
CK
DATA TOKEN
LO3
D R Q
CK
NOT DUPLICATE

FIG. 9(A)



00 aa cc 04 00 55 33 00 cc aa 33 00 04 08 00 04 55 00
00 aa cc 04 00 55 33 cc aa 33 00 04 08 04 55
00 aa cc 04 55 33 cc aa 33 00 04 08 04
143 208 273 338 403 468 533 598 663 728 793 858 923

FIG. 9(B)



PHØ
PH1
NOT_RESET
IN_EXTN
IN_DATA
IN_VALID
IN_ACCEPT
MID_EXTN
MID_DATA
MID_VALID
MID_ACCEPT
OUT_EXTN
OUT_DATA
OUT_VALID
OUT_ACCEPT
DATA_TOKEN
NOT_DUPLICATE
cc
00
33
aa
55
923
988
1053
1118
1183
1248
1313

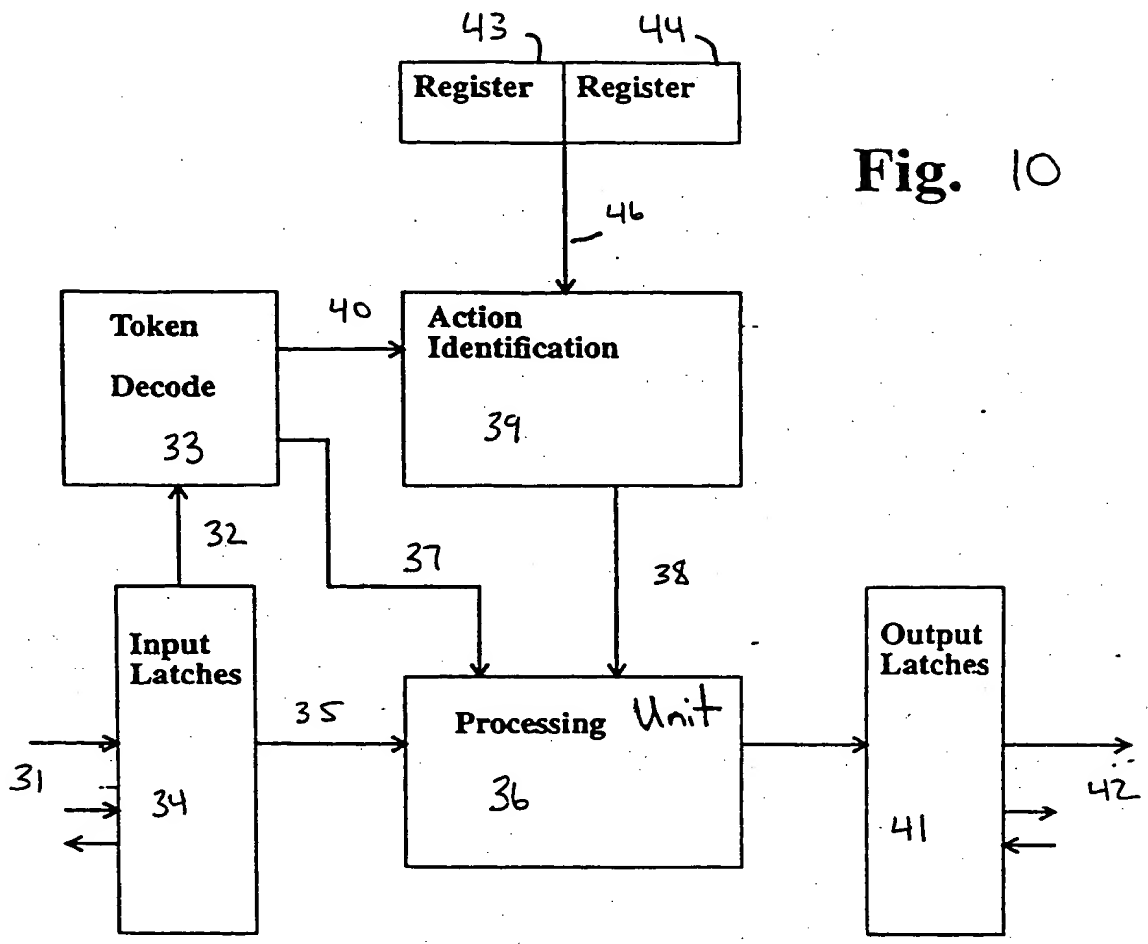
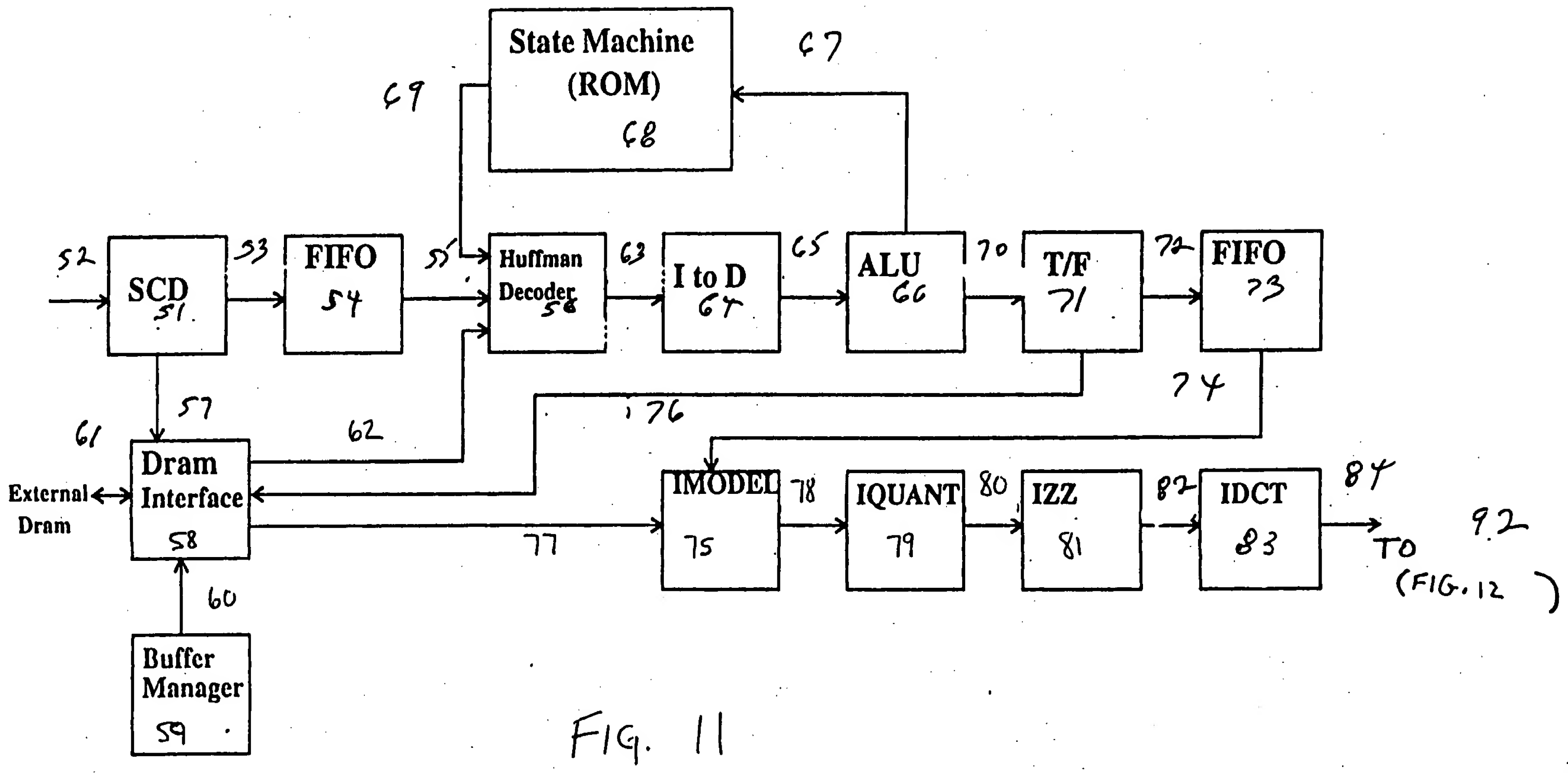

43
44
Register
Register
46
Token
Decode
33
40
Action
Identification
39
32
37
38
Input
Latches
34
31
35
Processing Unit
36
Output
Latches
41
42

Fig. 10



State Machine (ROM) 68
67
69
52
SCD 51
53
FIFO 54
55
Huffman Decoder 56
63
I to D 64
65
ALU 66
70
T/F 71
72
FIFO 73
74
76
57
61
62
External Dram
Dram Interface 58
60
Buffer Manager 59
77
IMODEL 75
78
IQUANT 79
80
IZZ 81
82
IDCT 83
84
TO (FIG. 12)
92

FIG. 11

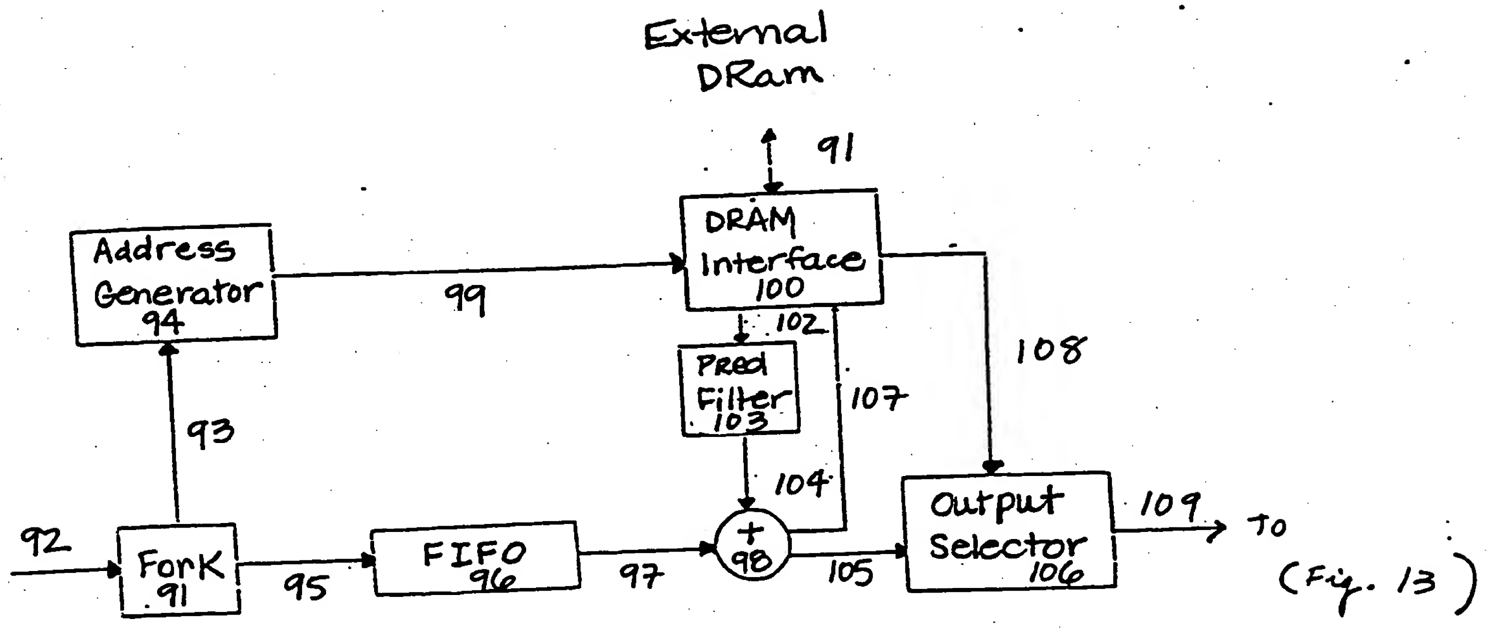

External
DRam
91
DRAM
Interface
100
Address
Generator
94
99
102
108
Pred
Filter
103
107
93
104
92
Fork
91
95
FIFO
96
97
+
98
105
Output
Selector
106
109
To
(Fig. 13)

Fig. 12

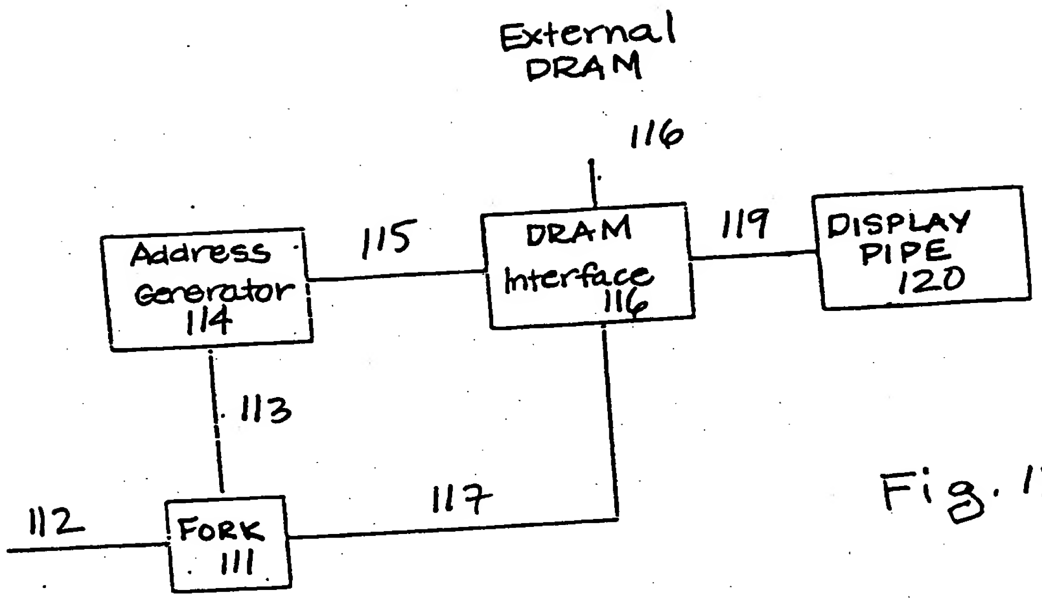

External DRAM
116
Address Generator 114
115
DRAM Interface 116
119
DISPLAY PIPE 120
113
112
FORK 111
117

Fig. 13

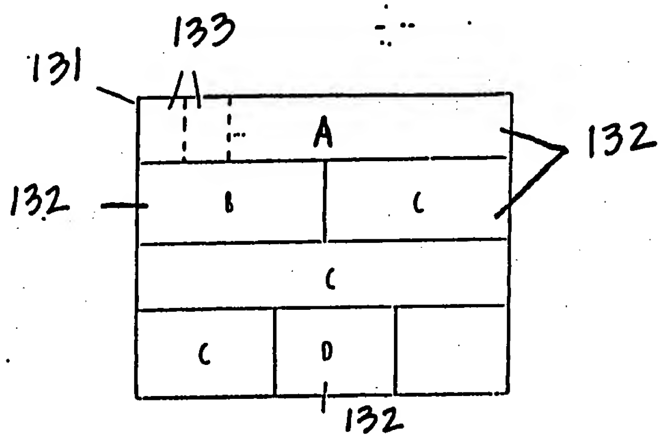


Fig. 14a

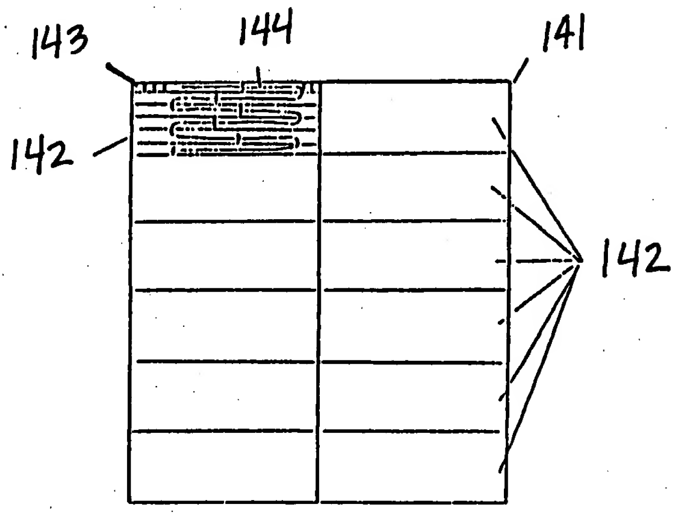


Fig. 14b

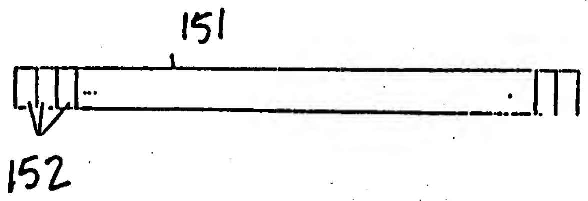


Fig. 14c

FIG. 15

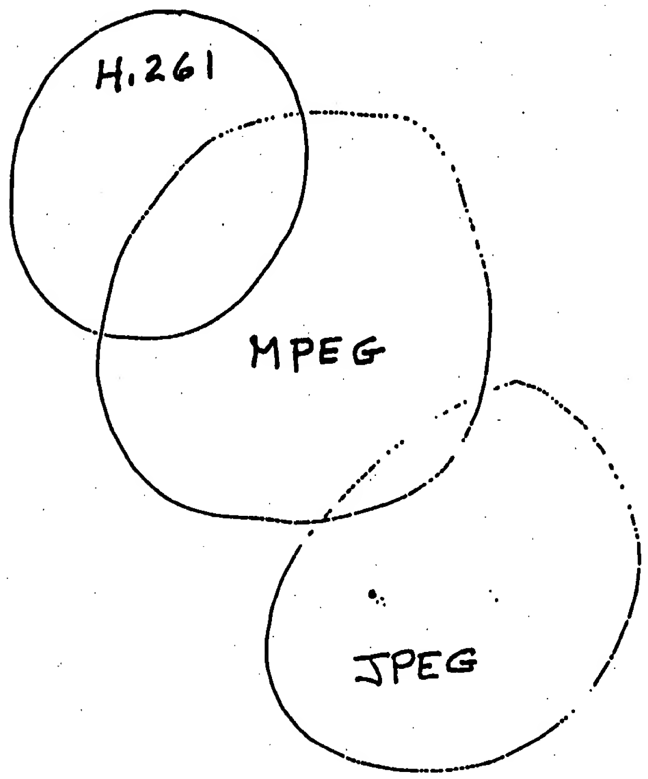


Fig. 16

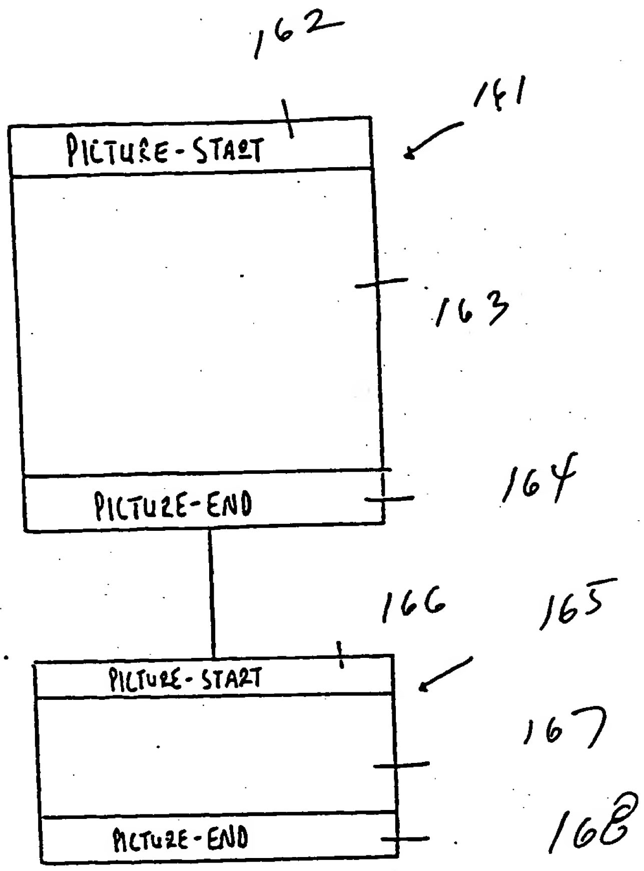

162
161
PICTURE-START
163
164
PICTURE-END
166
165
PICTURE-START
167
168
PICTURE-END

Fig. 17

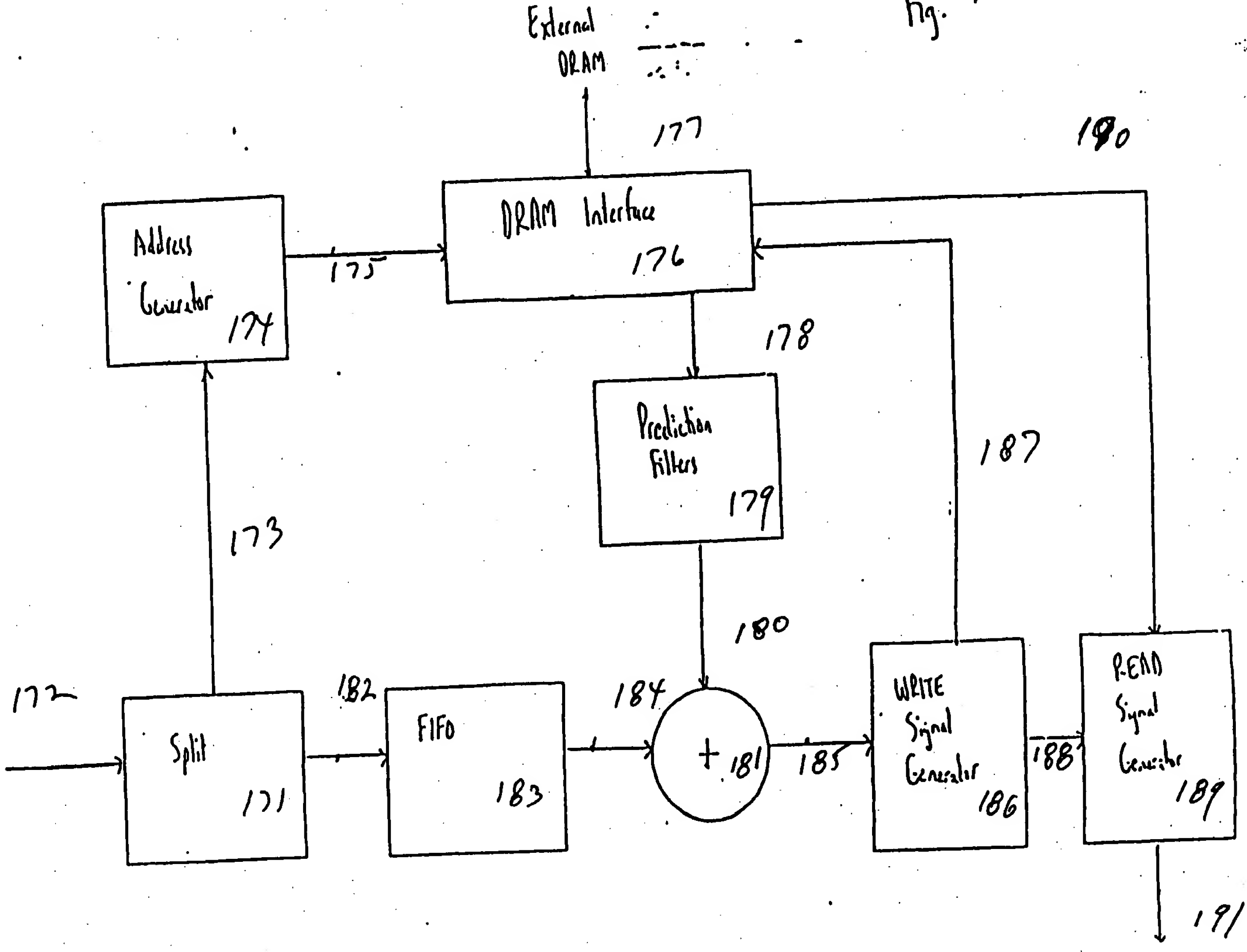

External DRAM
177
DRAM Interface 176
Address Generator 174
175
178
Prediction Filters 179
190
187
173
180
172
Split 171
182
FIFO 183
184
+ 181
185
WRITE Signal Generator 186
188
READ Signal Generator 189
191

Fig. 18

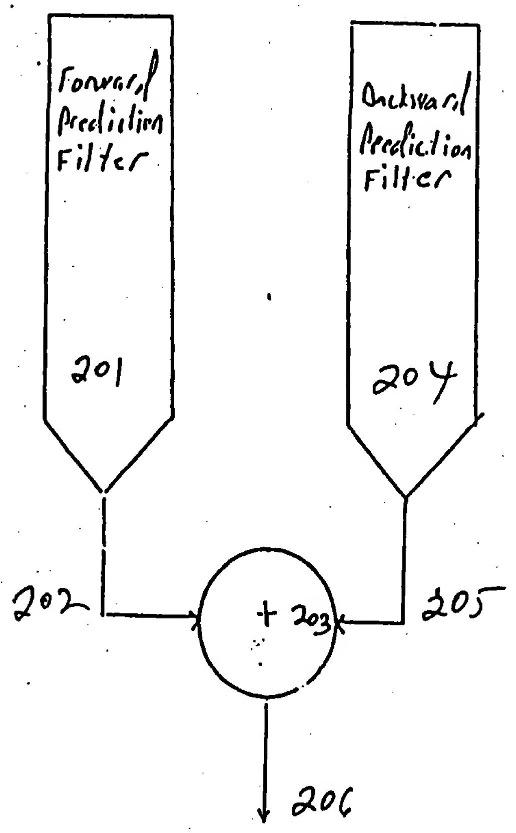

Forward Prediction Filter
201
Backward Prediction Filter
204
202
+
203
205
206

Fig. 19

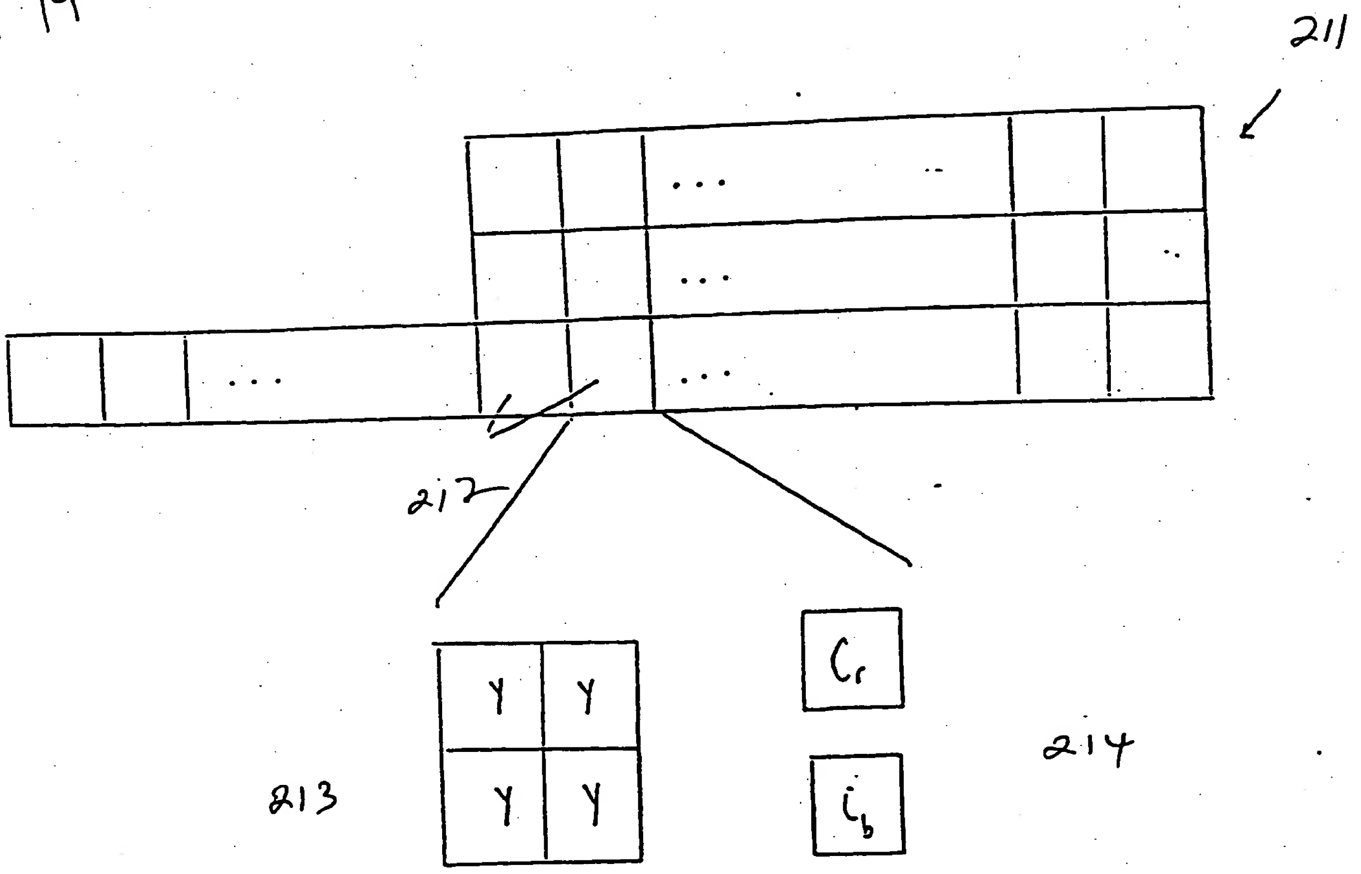
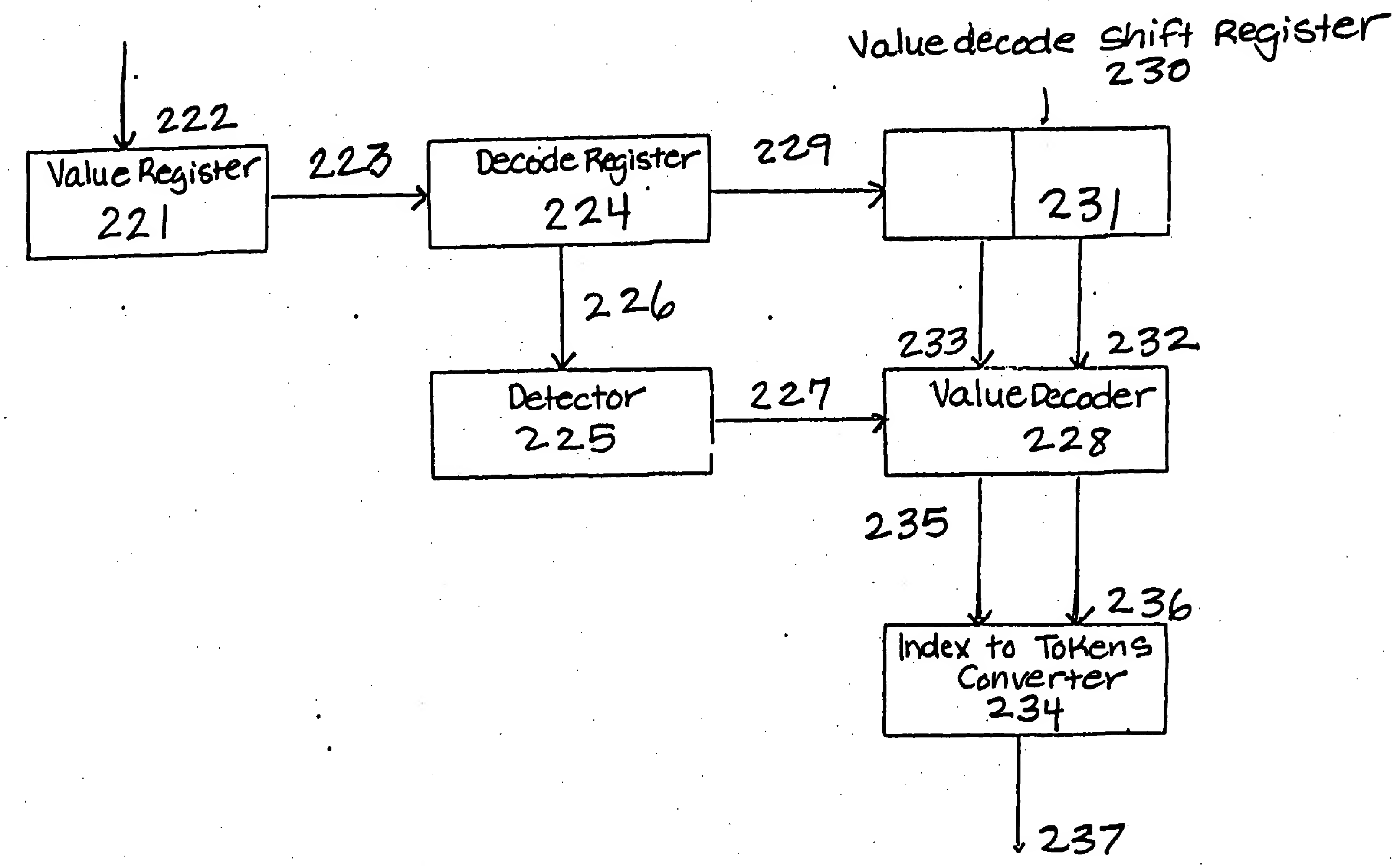


Fig. 20



Value decode shift Register 230
222
Value Register 221
223
Decode Register 224
229
231
226
233
232
Detector 225
227
Value Decoder 228
235
236
Index to Tokens Converter 234
237

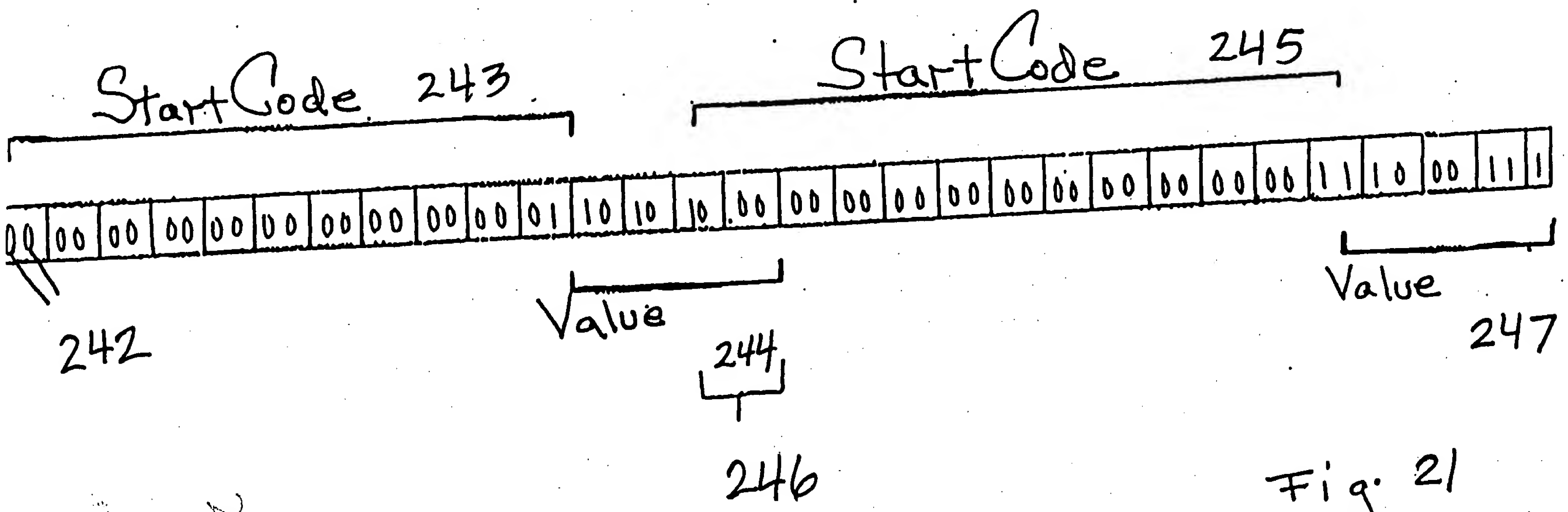
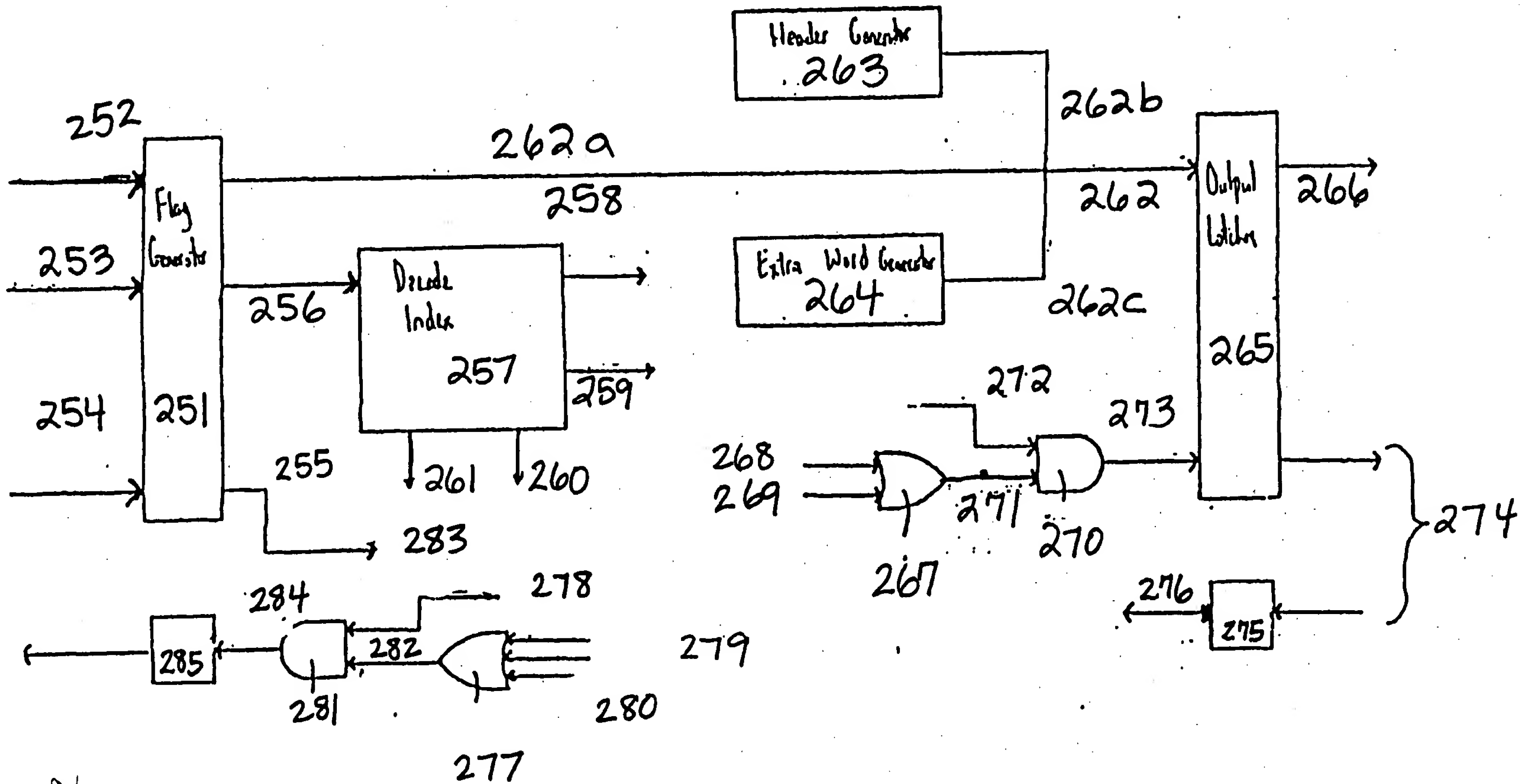

Start Code 243
Start Code 245
00 00 00 00 00 00 00 00 00 00 01 10 10 10 00 00 00 00 00 00 00 00 00 00 00 11 10 00 11 1
Value
Value
242
244
246
247

Fig. 21

Fig. 22



252
253
254
Flag Generator
251
262a
258
256
Decode Index
257
259
261
260
255
283
Header Generator 263
Extra Word Generator 264
262b
262
262c
Output Latches
265
266
268
269
267
271
272
270
273
274
276
275
278
279
280
277
282
284
281
285

29

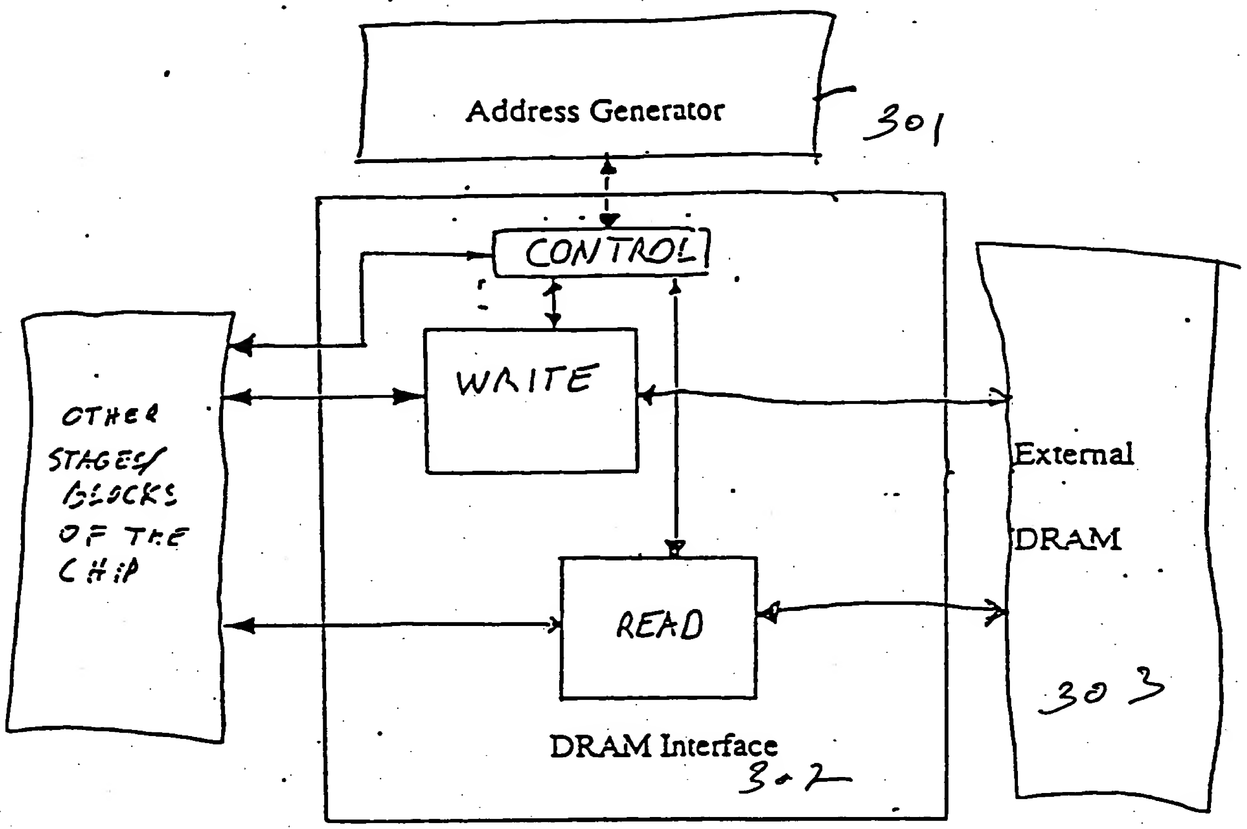

Address Generator
301
CONTROL
WRITE
READ
OTHER STAGES/ BLOCKS OF THE CHIP
External DRAM
303
DRAM Interface
302

FIG. 23

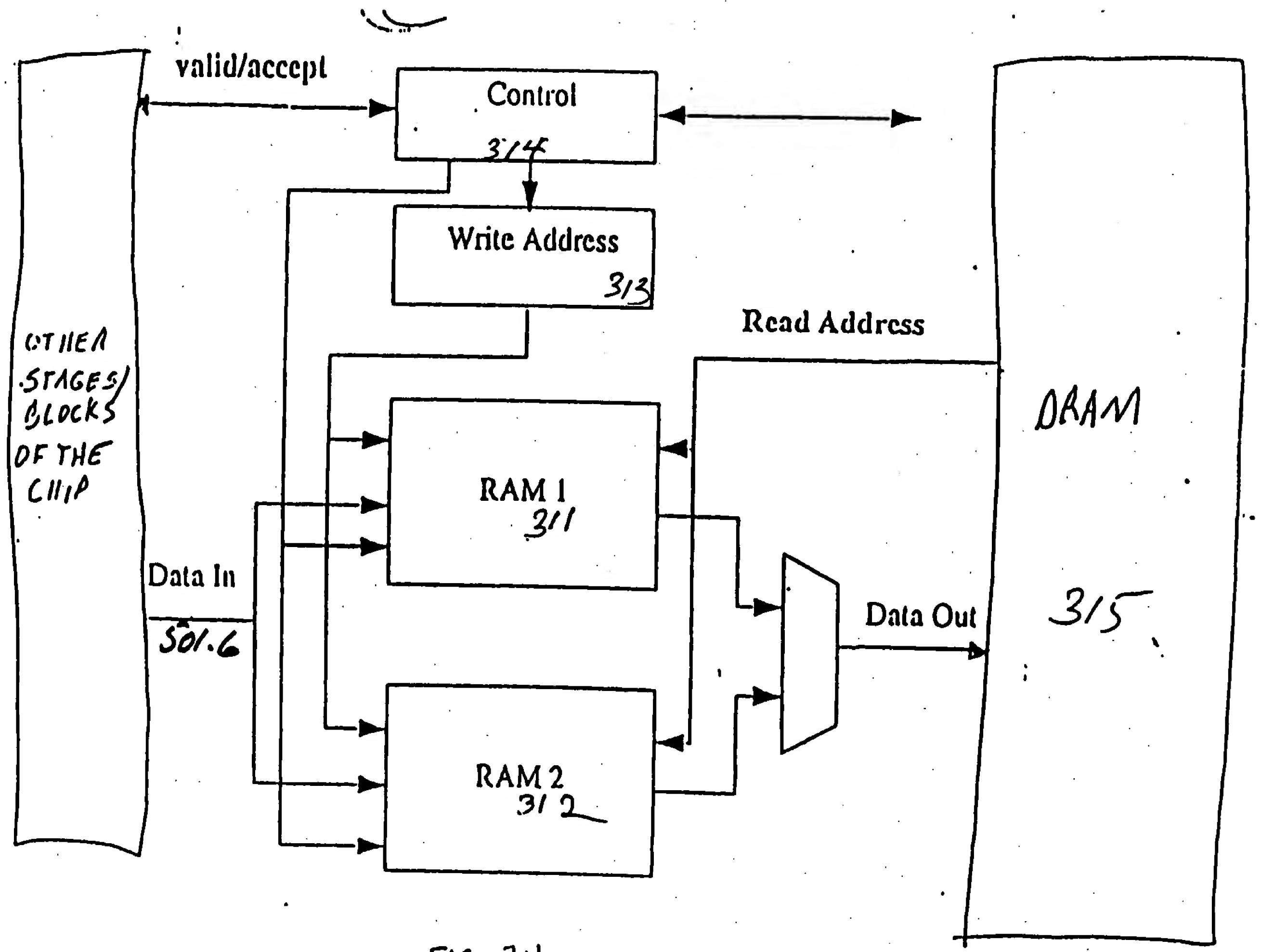

valid/accept
Control
314
Write Address
313
Read Address
OTHER STAGES/BLOCKS OF THE CHIP
RAM 1
311
RAM 2
312
Data In
501.6
Data Out
DRAM
315

FIG. 24

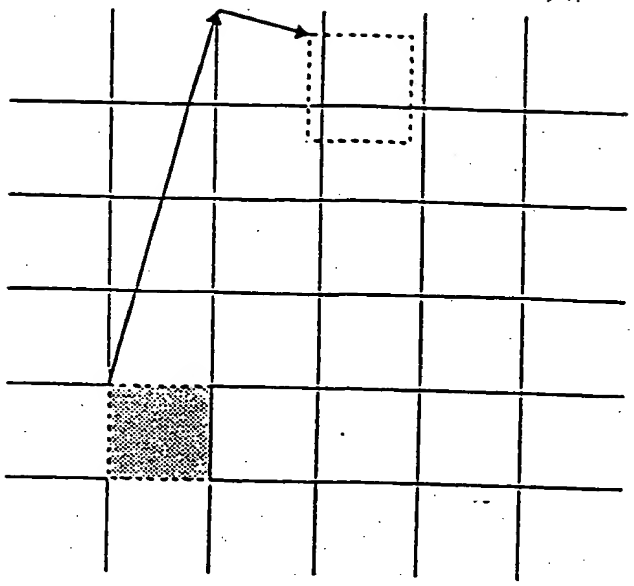


FIG. 25

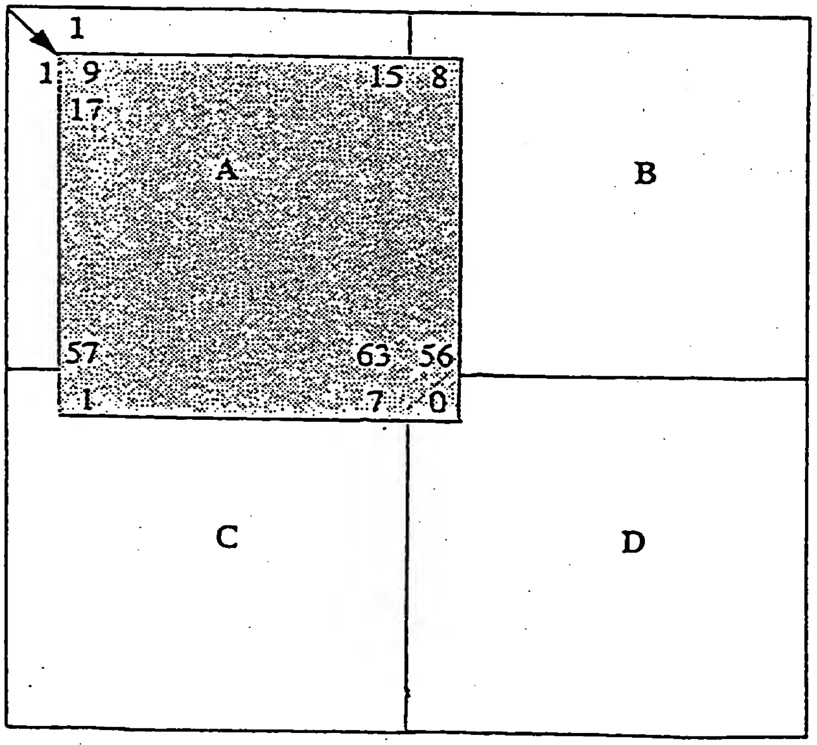


FIG. 26

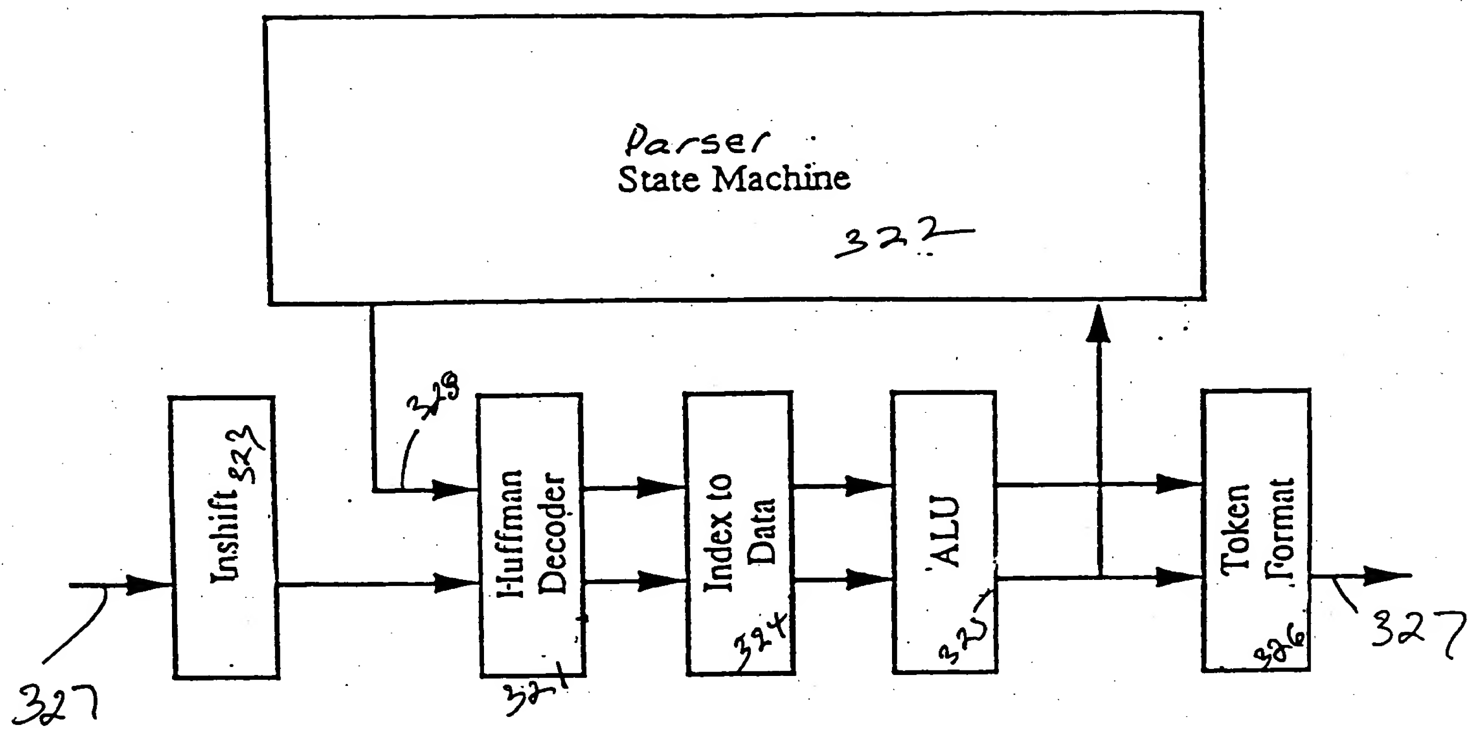

Parser
State Machine
322
Inshift
323
328
Huffman Decoder
321
Index to Data
324
ALU
325
Token Format
326
327
327

FIG. 27

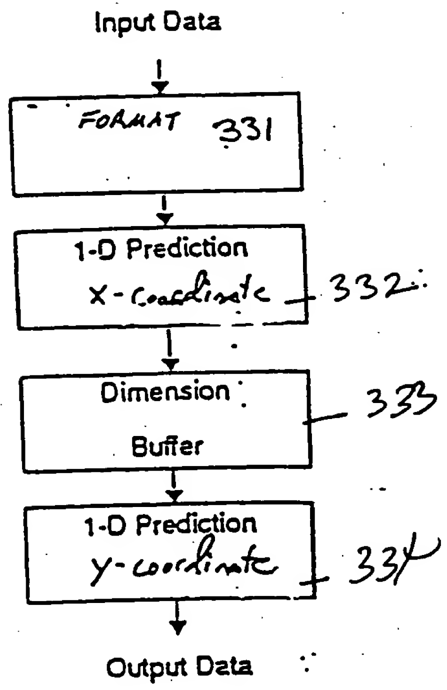

Input Data
FORMAT 331
1-D Prediction
x-coordinate
332
Dimension
Buffer
333
1-D Prediction
y-coordinate
334
Output Data

FIG. 28

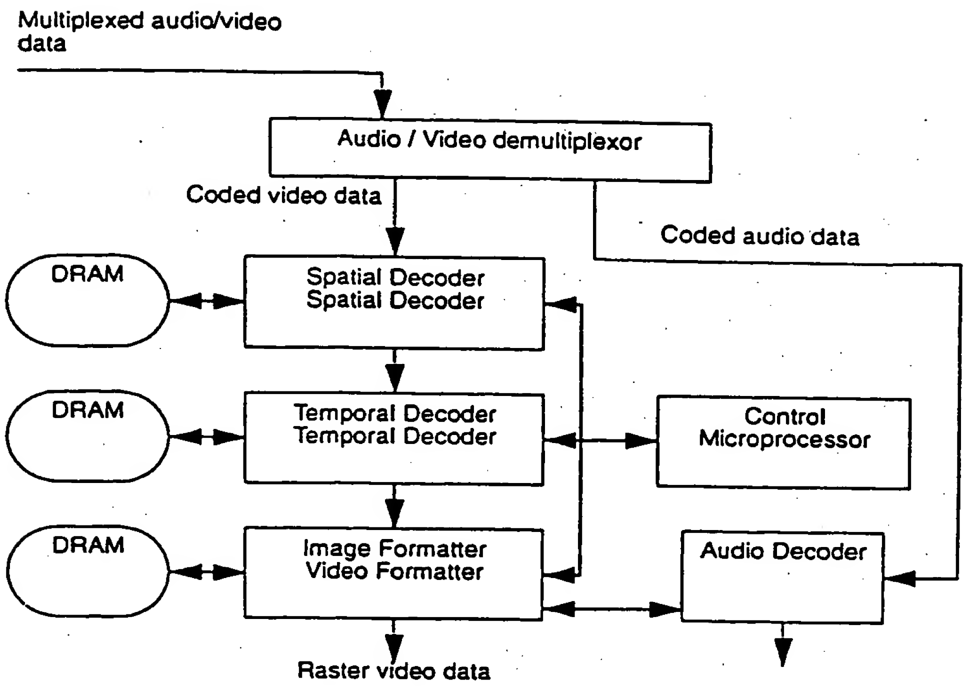

Multiplexed audio/video data
Audio / Video demultiplexor
Coded video data
Coded audio data
DRAM
Spatial Decoder
Spatial Decoder
DRAM
Temporal Decoder
Temporal Decoder
Control Microprocessor
DRAM
Image Formatter
Video Formatter
Audio Decoder
Raster video data

Figure 29

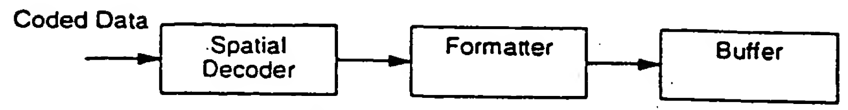


Figure 30

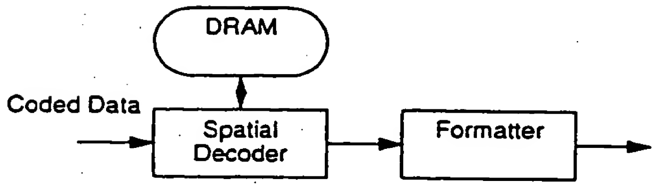


Figure 31

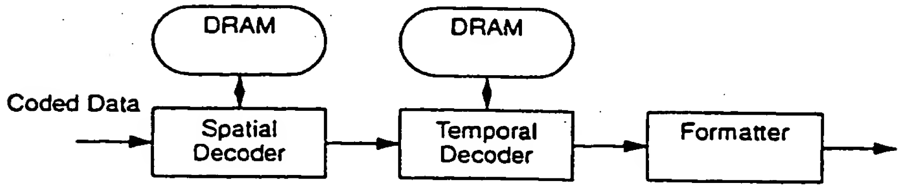


Figure 32

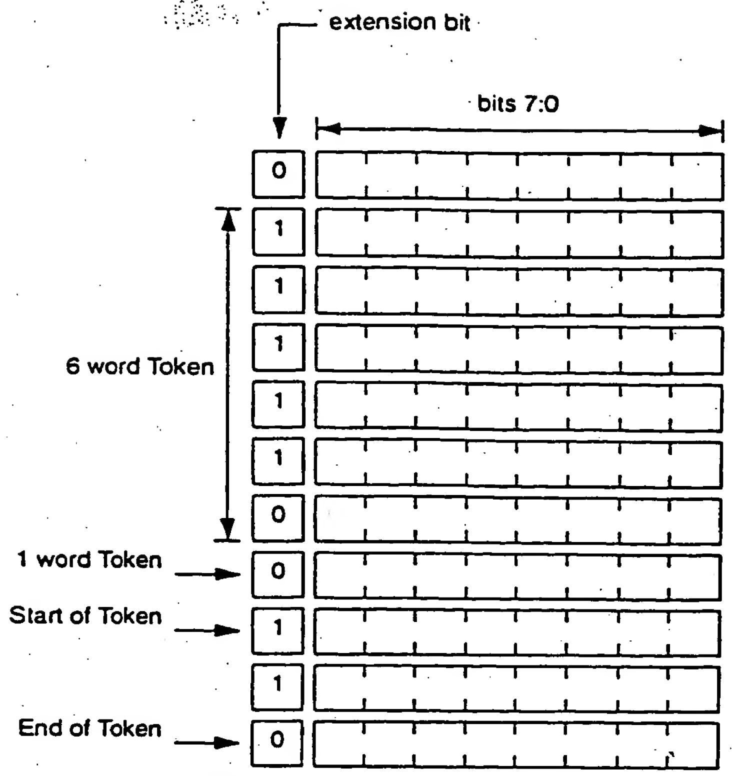


Figure 33

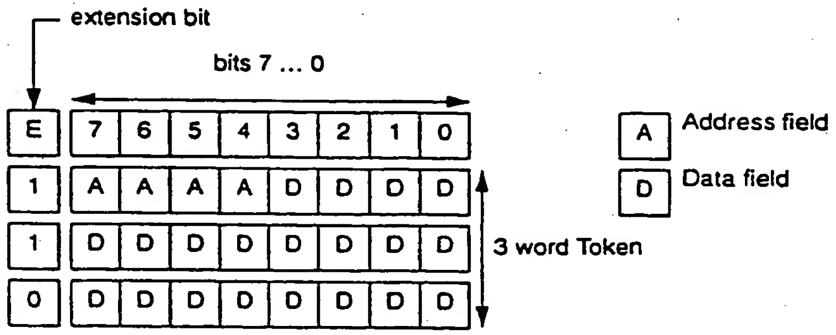


Figure 34

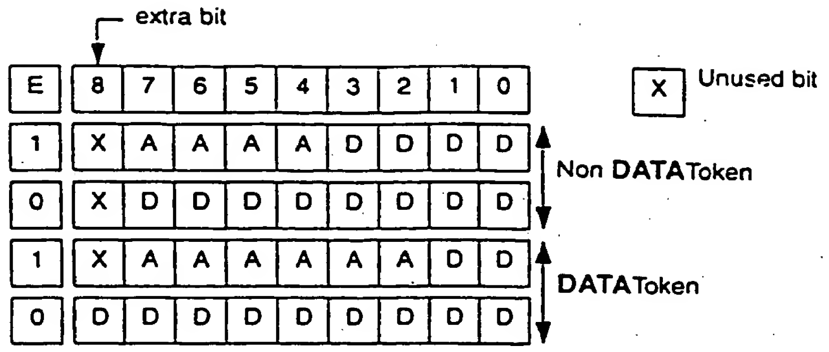


Figure 35

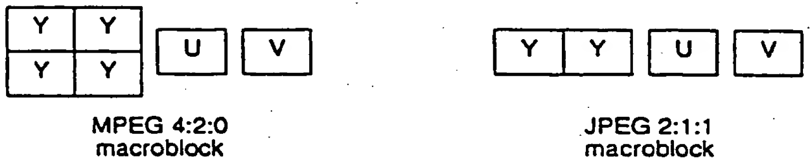


Figure 36

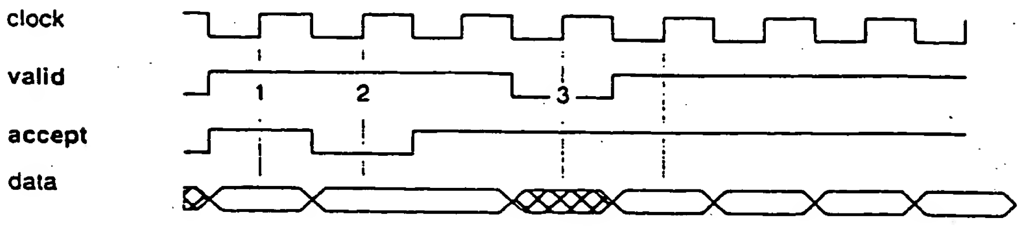


Figure 37

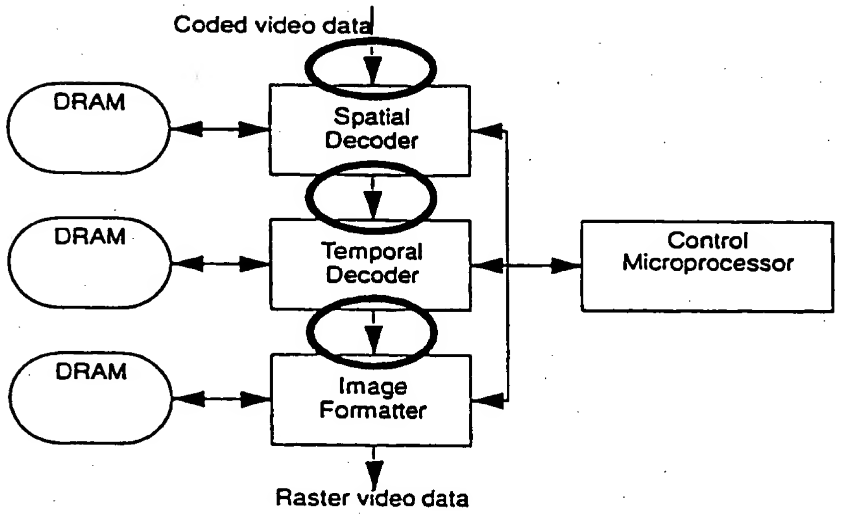


Figure 38

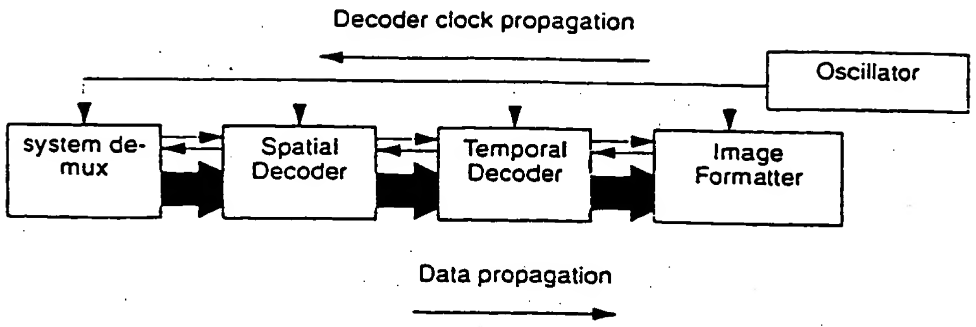


Figure 39

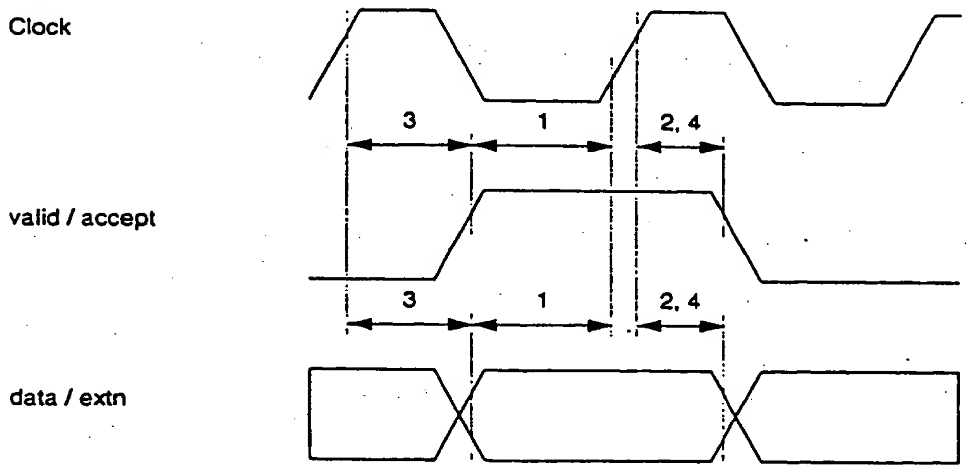


Figure 40

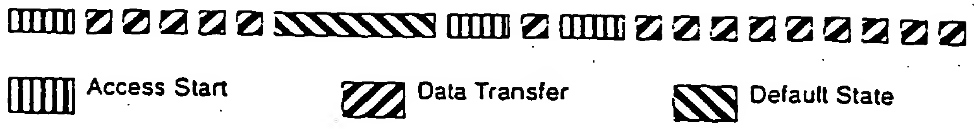


Figure 41

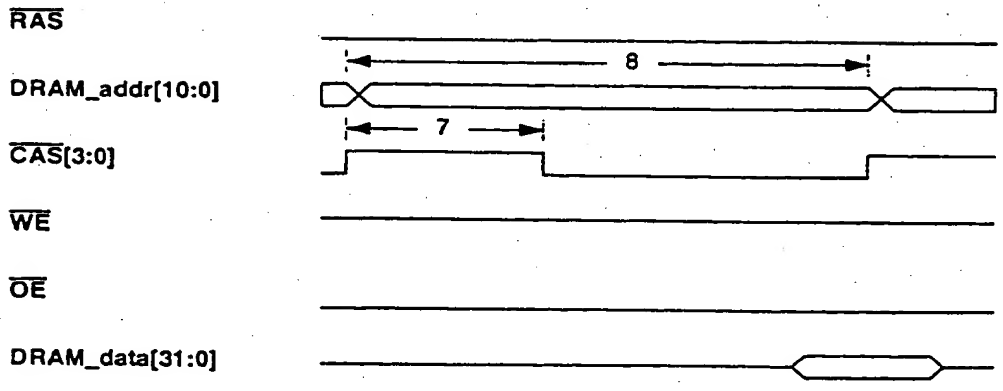


Figure 42

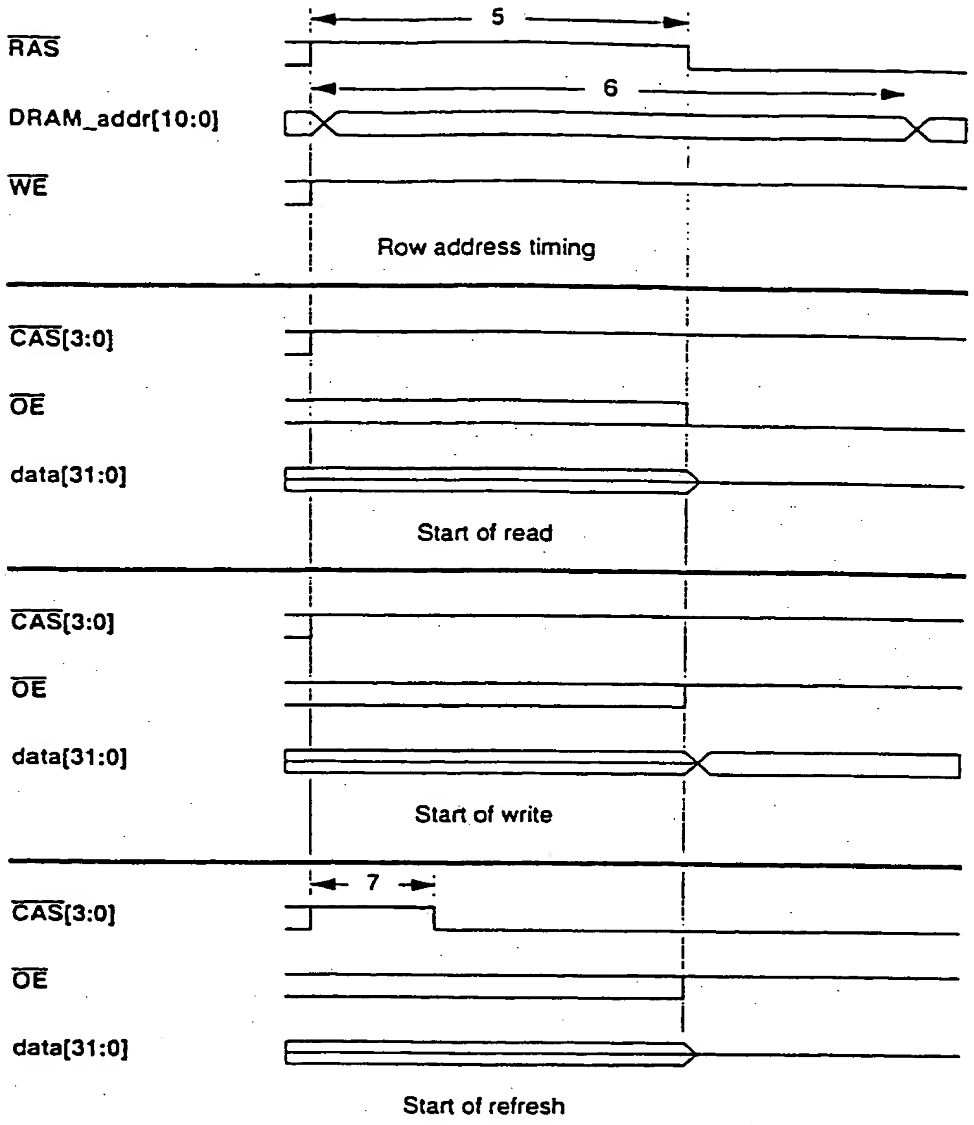

5
6
7
$\overline{RAS}$
DRAM_addr[10:0]
$\overline{WE}$
Row address timing
$\overline{CAS}$[3:0]
$\overline{OE}$
data[31:0]
Start of read
$\overline{CAS}$[3:0]
$\overline{OE}$
data[31:0]
Start of write
$\overline{CAS}$[3:0]
$\overline{OE}$
data[31:0]
Start of refresh

Figure 43

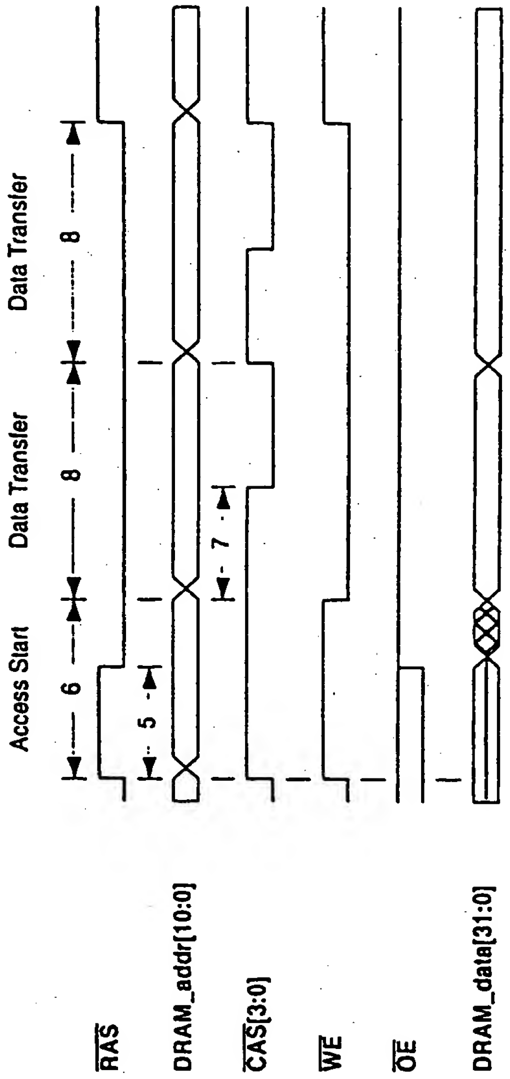

Access Start
6
Data Transfer
8
Data Transfer
8
5
7
$\overline{RAS}$
DRAM_addr[10:0]
$\overline{CAS}$[3:0]
$\overline{WE}$
$\overline{OE}$
DRAM_data[31:0]

Figure 44

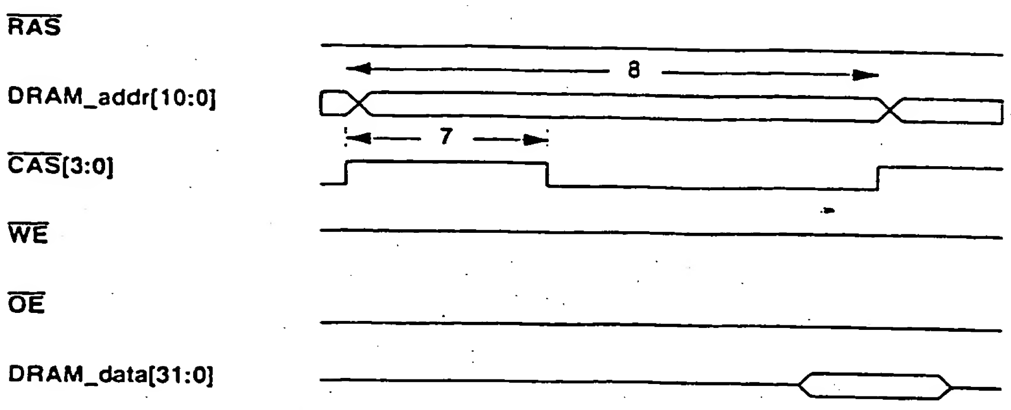


Figure 45

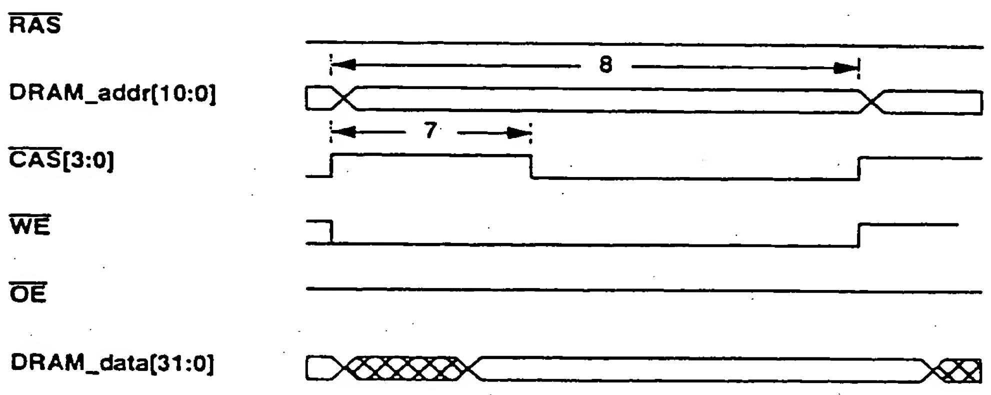


Figure 46

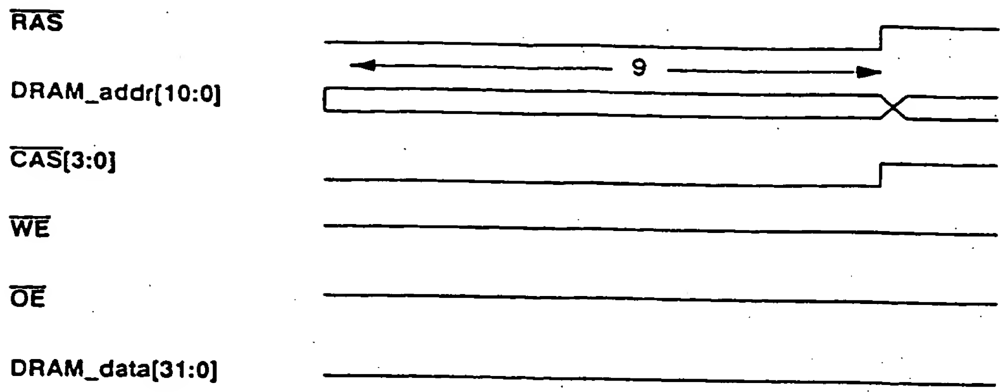


Figure 47

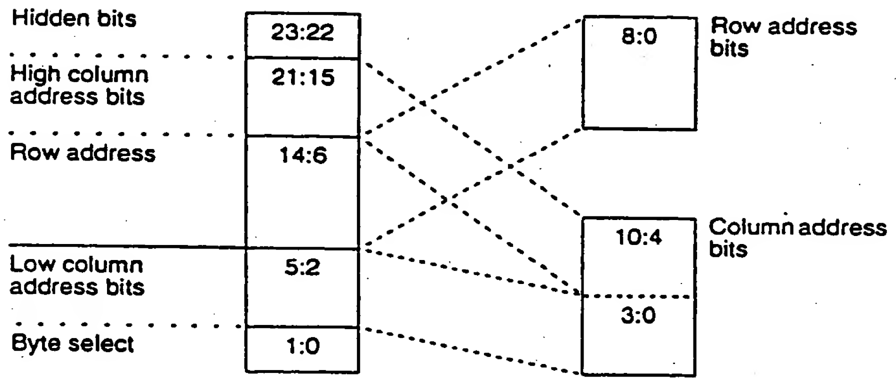


Figure 48

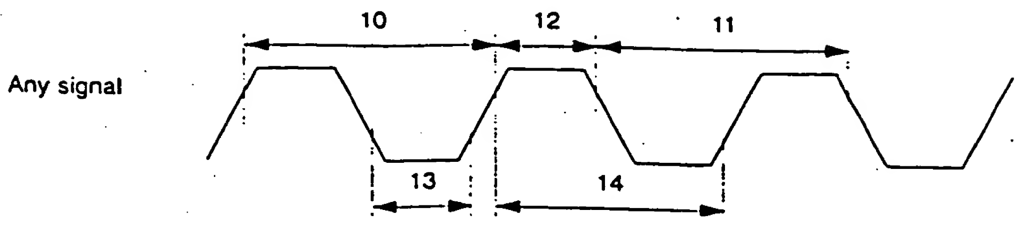


Figure 49

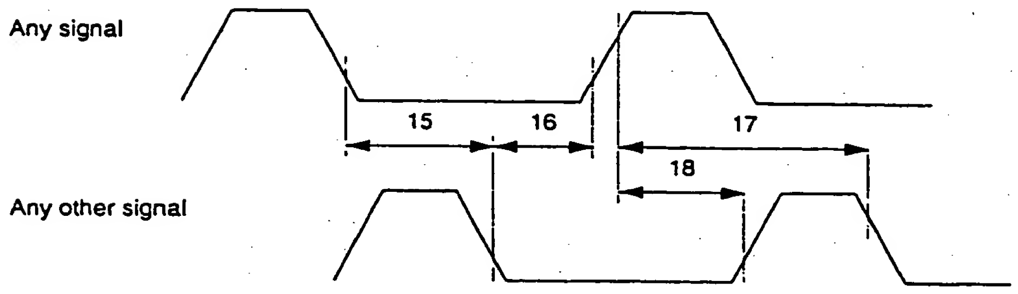


Figure 50

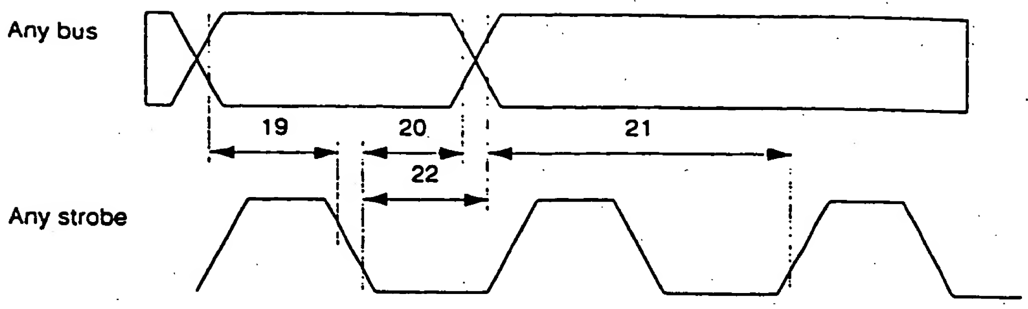


Figure . 51

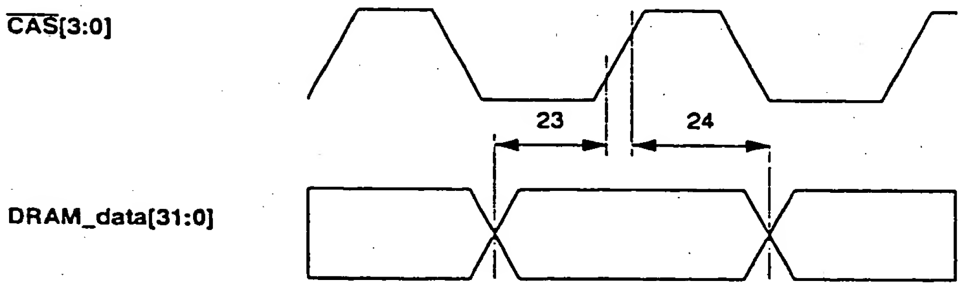


Figure 52

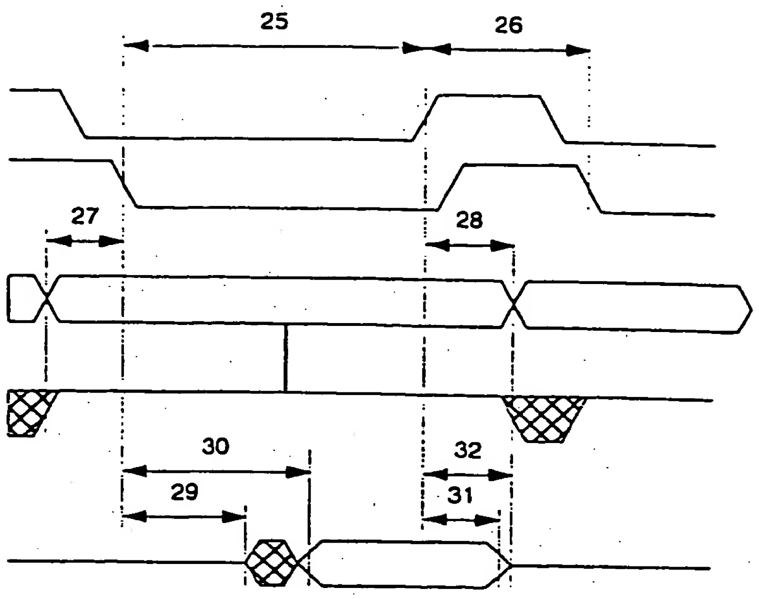


Figure 53

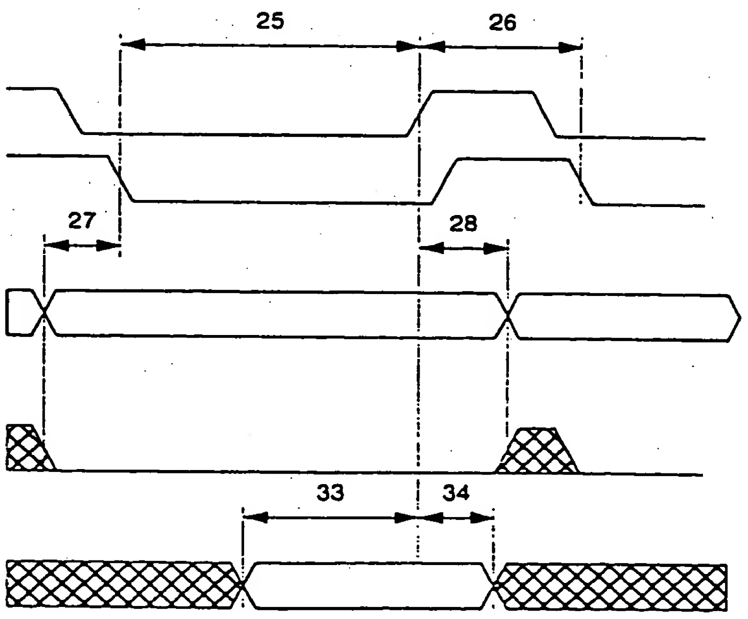


Figure 54

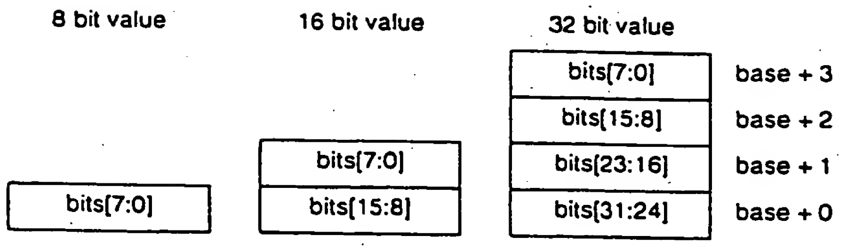

8 bit value
16 bit value
32 bit value
bits[7:0]
bits[7:0]
bits[15:8]
bits[7:0]
bits[15:8]
bits[23:16]
bits[31:24]
base + 3
base + 2
base + 1
base + 0

Figure 55

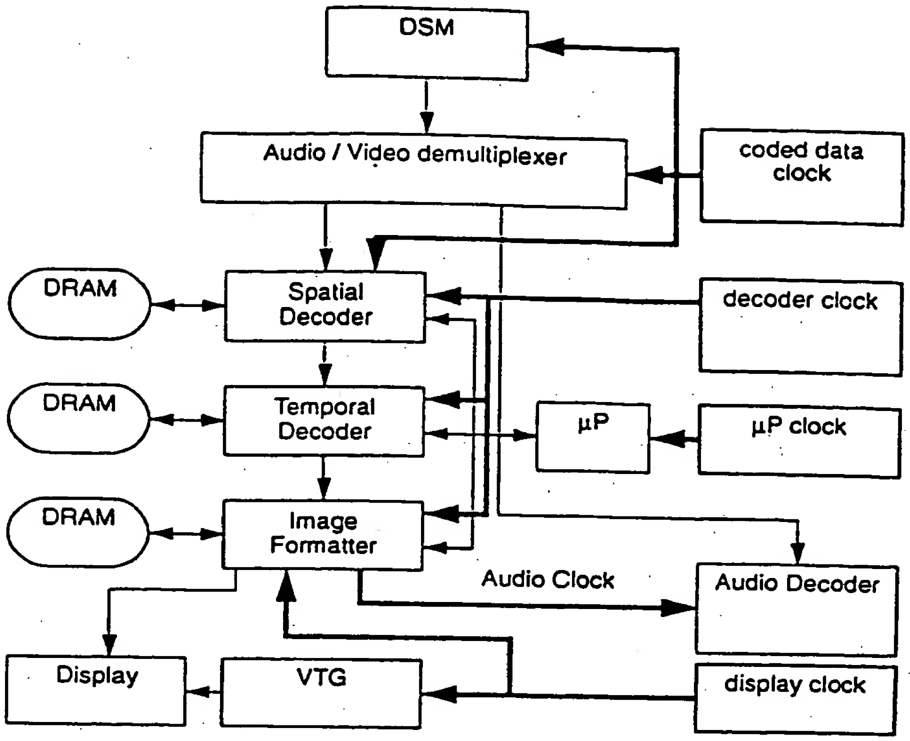


Figure 56

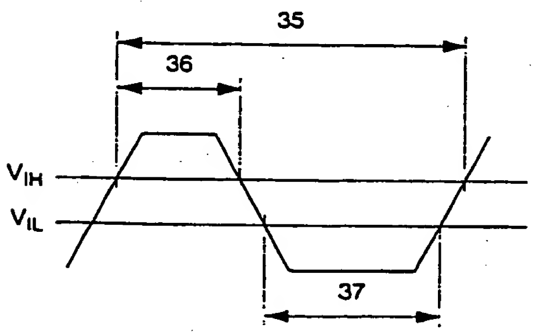
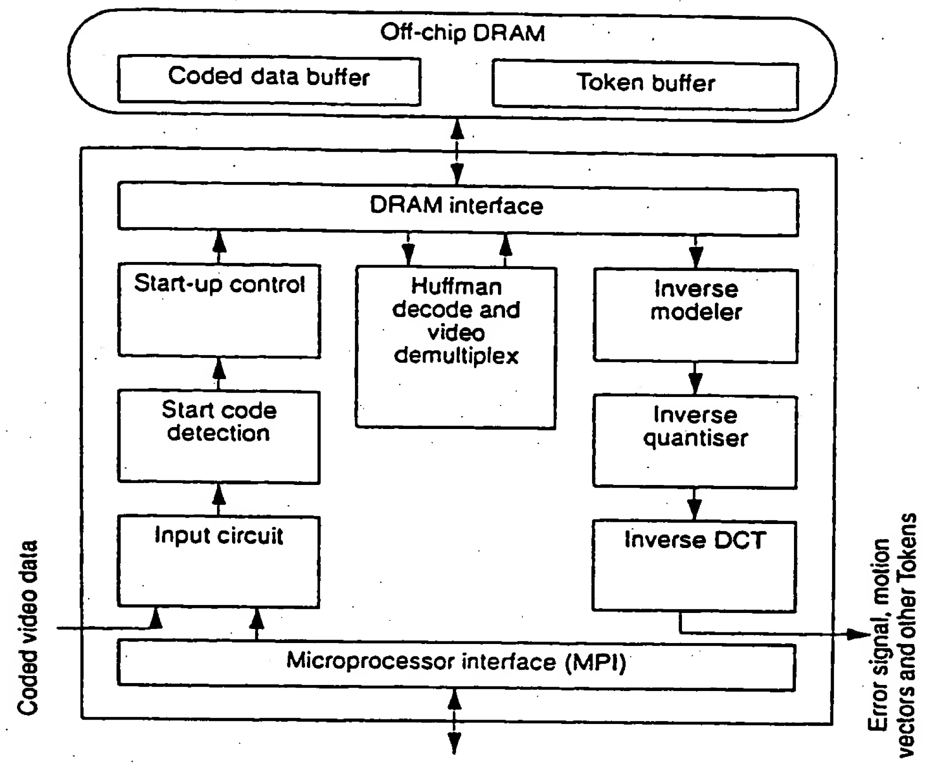
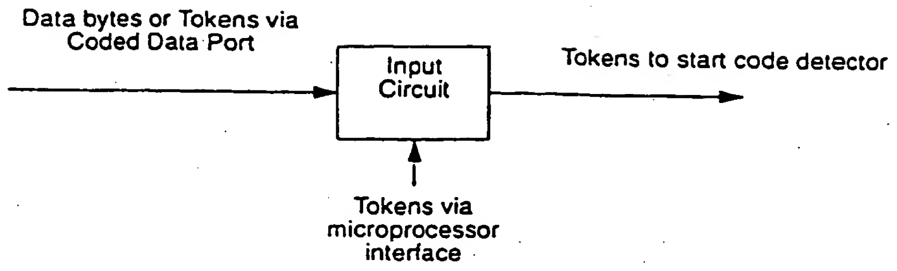


Figure 57



Off-chip DRAM
Coded data buffer
Token buffer
DRAM interface
Start-up control
Huffman decode and video demultiplex
Inverse modeler
Start code detection
Inverse quantiser
Input circuit
Inverse DCT
Coded video data
Microprocessor interface (MPI)
Error signal, motion vectors and other Tokens

Figure 58



Data bytes or Tokens via Coded Data Port
Input Circuit
Tokens to start code detector
Tokens via microprocessor interface

Figure 59

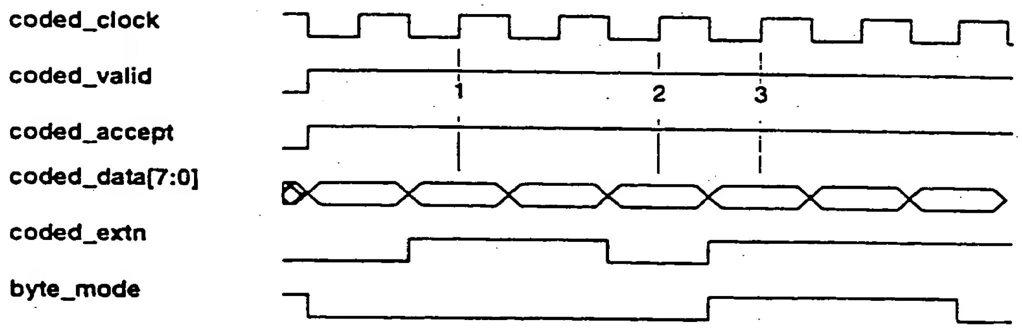


Figure 60

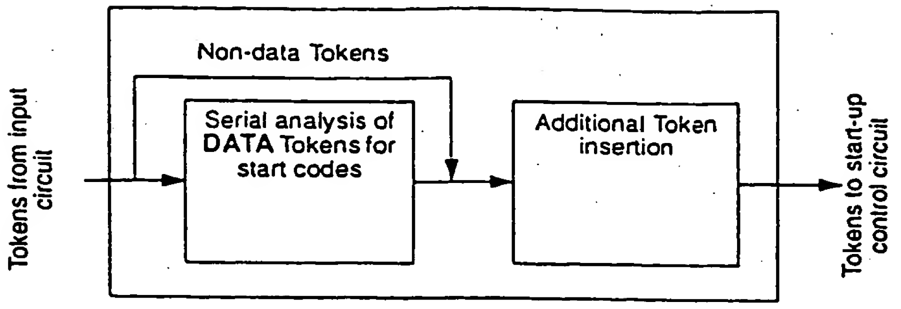

Non-data Tokens
Serial analysis of DATA Tokens for start codes
Additional Token insertion
Tokens from input circuit
Tokens to start-up control circuit

Figure 61

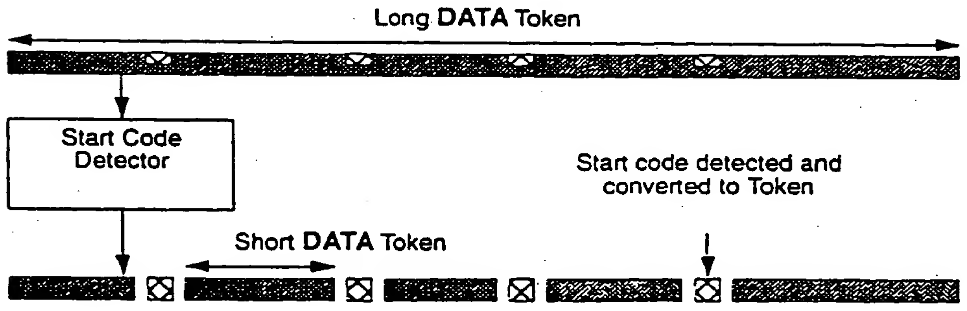


Figure 62

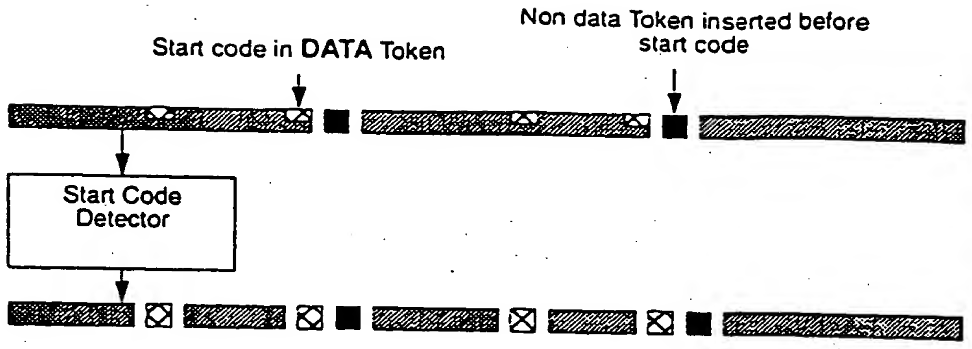


Figure 63

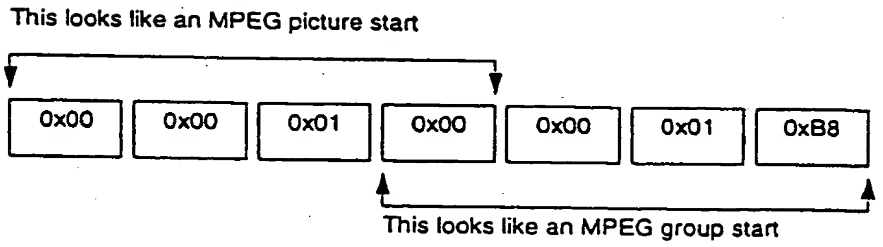

This looks like an MPEG picture start
0x00
0x00
0x01
0x00
0x00
0x01
0xB8
This looks like an MPEG group start

Figure 64

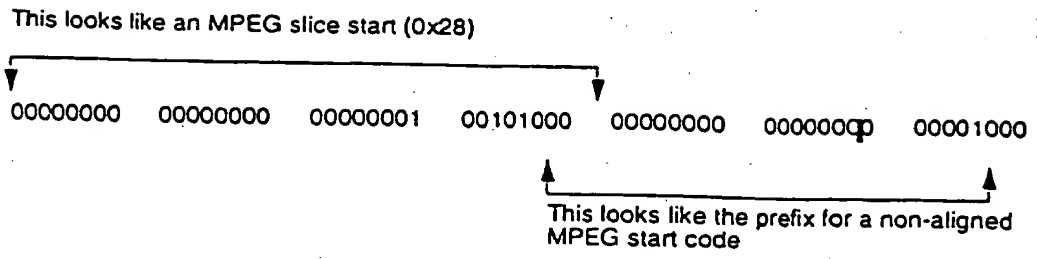

This looks like an MPEG slice start (0x28)
00000000 00000000 00000001 00101000 00000000 00000000 00001000
This looks like the prefix for a non-aligned MPEG start code

Figure 65

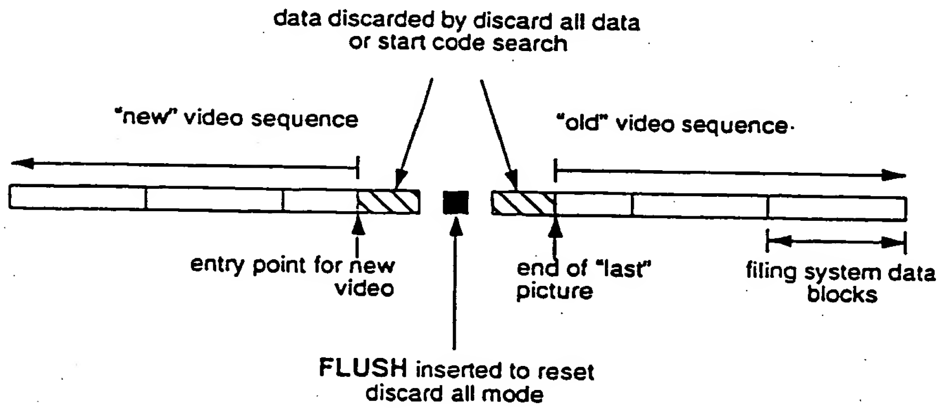

data discarded by discard all data or start code search
"new" video sequence
"old" video sequence
entry point for new video
end of "last" picture
filing system data blocks
FLUSH inserted to reset discard all mode

Figure 66

55

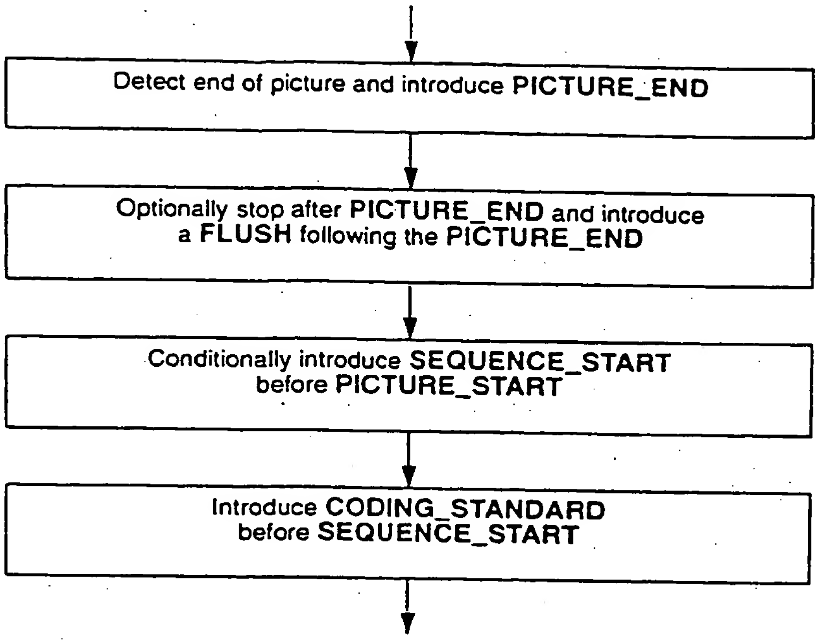

Detect end of picture and introduce PICTURE_END
Optionally stop after PICTURE_END and introduce a FLUSH following the PICTURE_END
Conditionally introduce SEQUENCE_START before PICTURE_START
Introduce CODING_STANDARD before SEQUENCE_START

Figure. 67

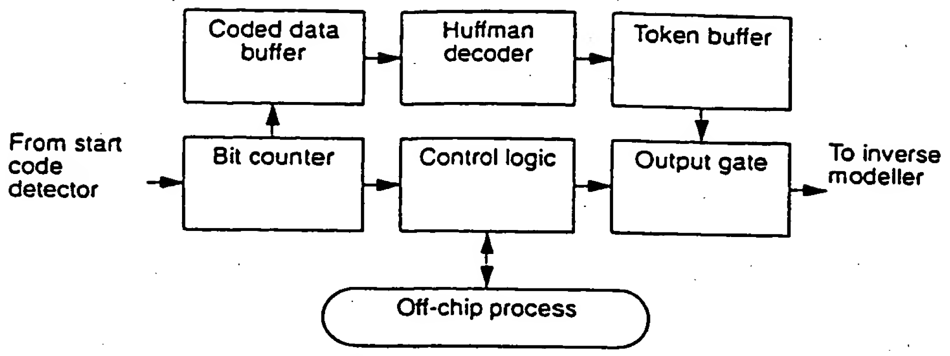


Figure 68

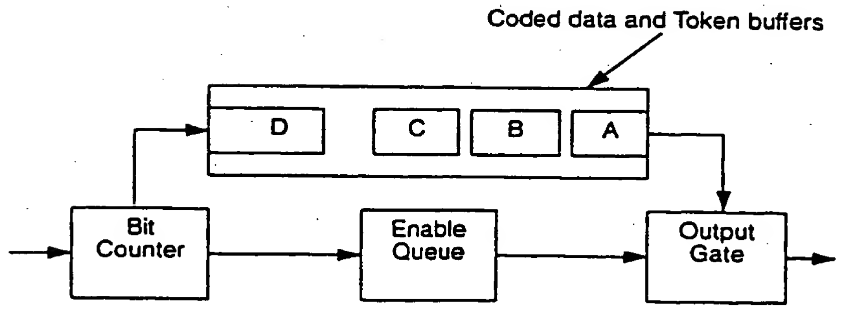


Figure. 69

57

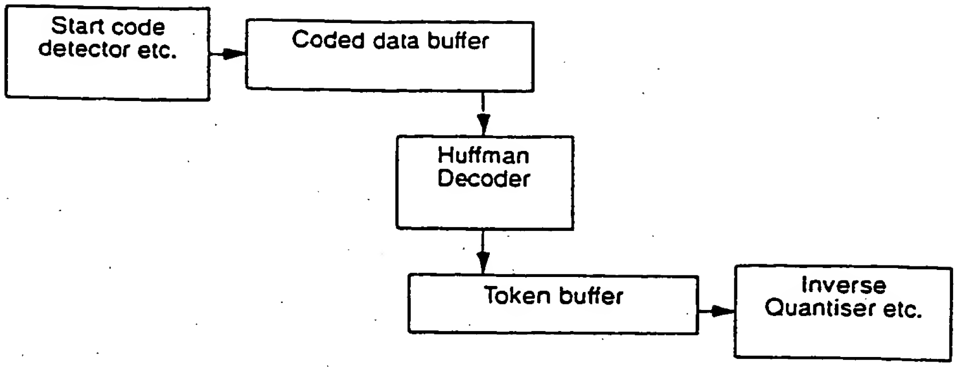


Figure 70

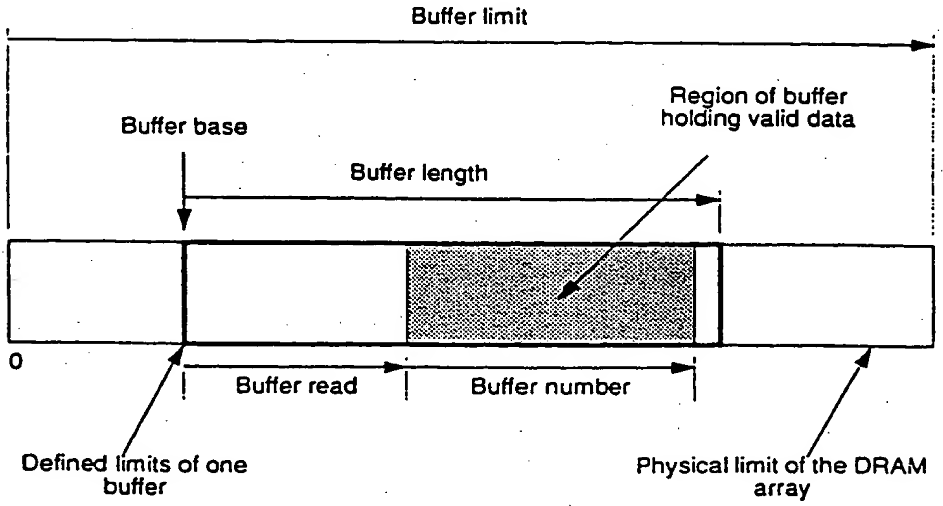
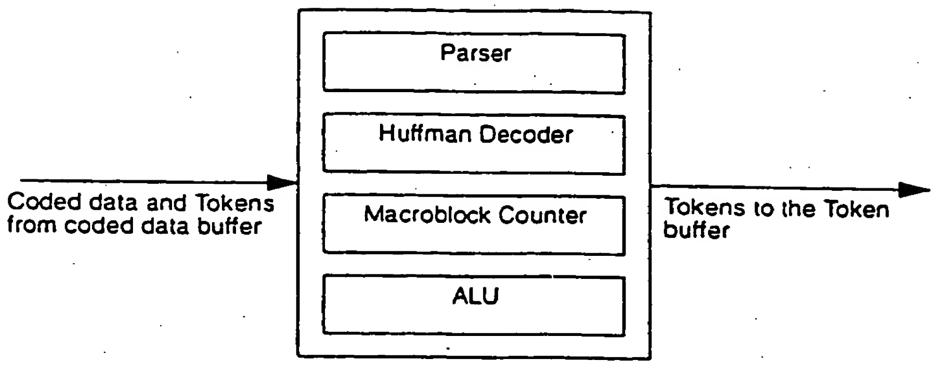

Buffer limit
Region of buffer holding valid data
Buffer base
Buffer length
0
Buffer read
Buffer number
Defined limits of one buffer
Physical limit of the DRAM array

Figure 71



Parser
Huffman Decoder
Macroblock Counter
ALU
Coded data and Tokens from coded data buffer
Tokens to the Token buffer

Figure 72

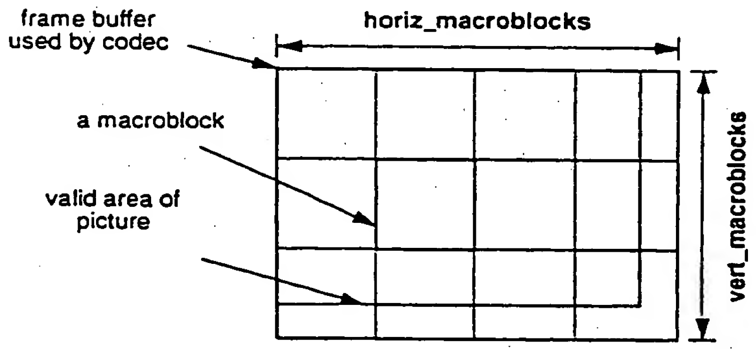


Figure. 73

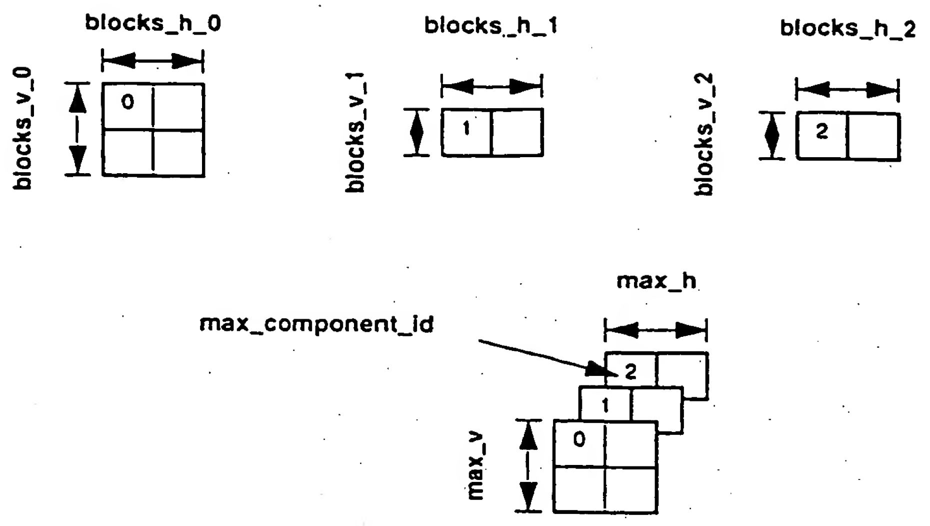


Figure 74

$$\text{horiz\_macroblocks} = \frac{\text{horiz\_pels} + 15}{16}$$

$$\text{vert\_macroblocks} = \frac{\text{vert\_pels} + 15}{16}$$

Figure 75

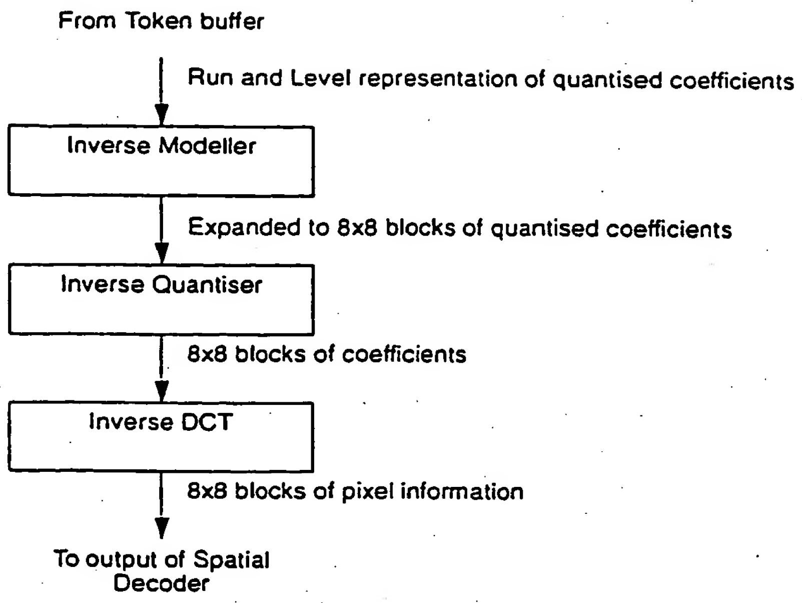

From Token buffer
Run and Level representation of quantised coefficients
Inverse Modeller
Expanded to 8x8 blocks of quantised coefficients
Inverse Quantiser
8x8 blocks of coefficients
Inverse DCT
8x8 blocks of pixel information
To output of Spatial Decoder

Figure 76

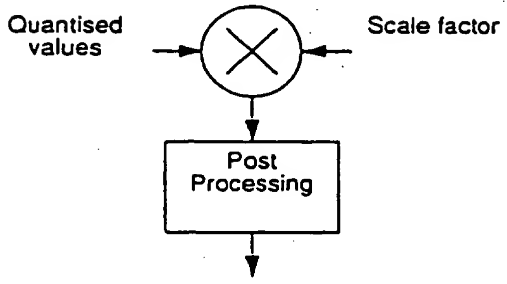


Figure . 77

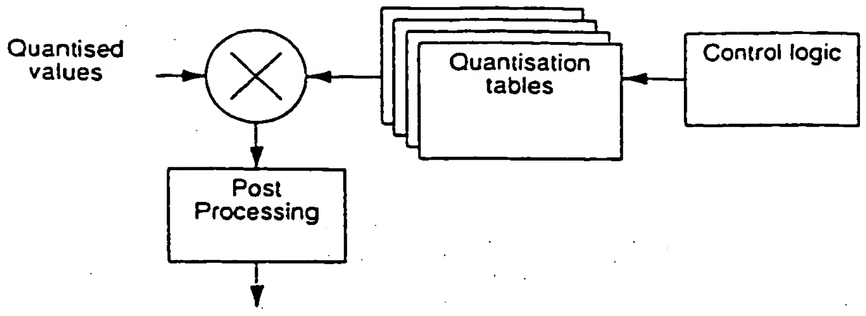


Figure 78

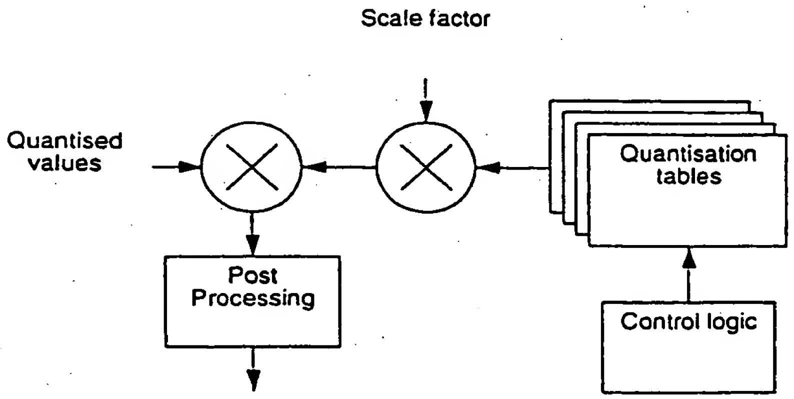


Figure 79

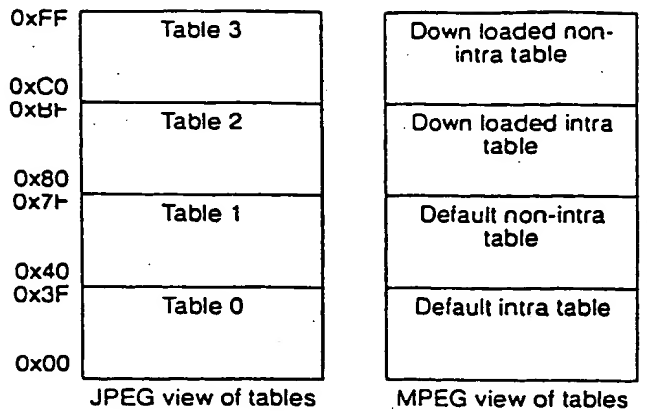

0xFF
0xC0
0xBF
0x80
0x7F
0x40
0x3F
0x00
Table 3
Table 2
Table 1
Table 0
JPEG view of tables
Down loaded non-intra table
Down loaded intra table
Default non-intra table
Default intra table
MPEG view of tables

Figure. 80

(67)

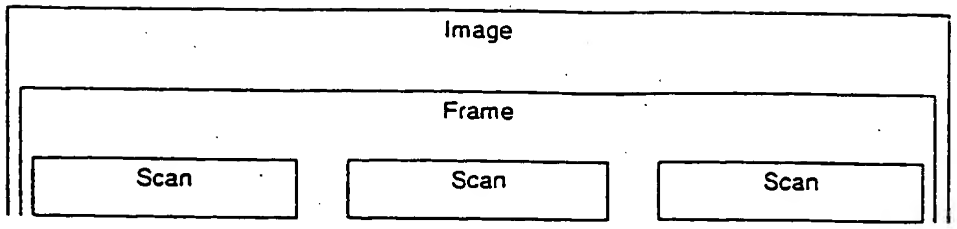


Figure 81

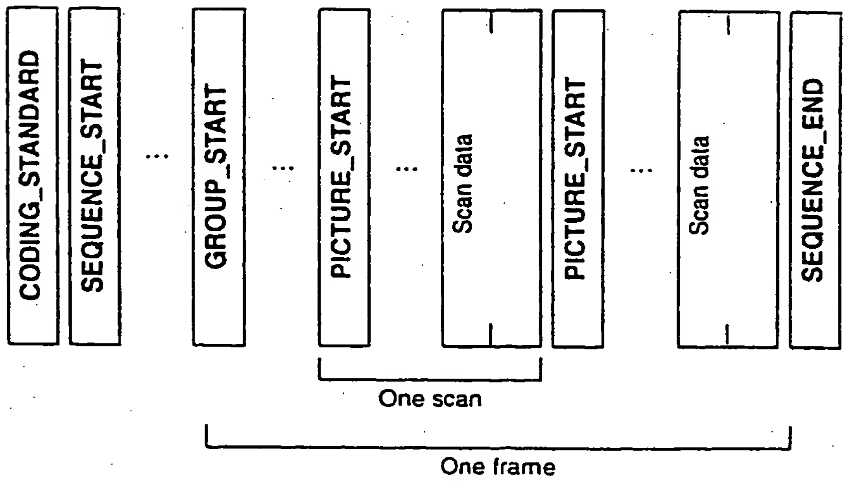


Figure 82

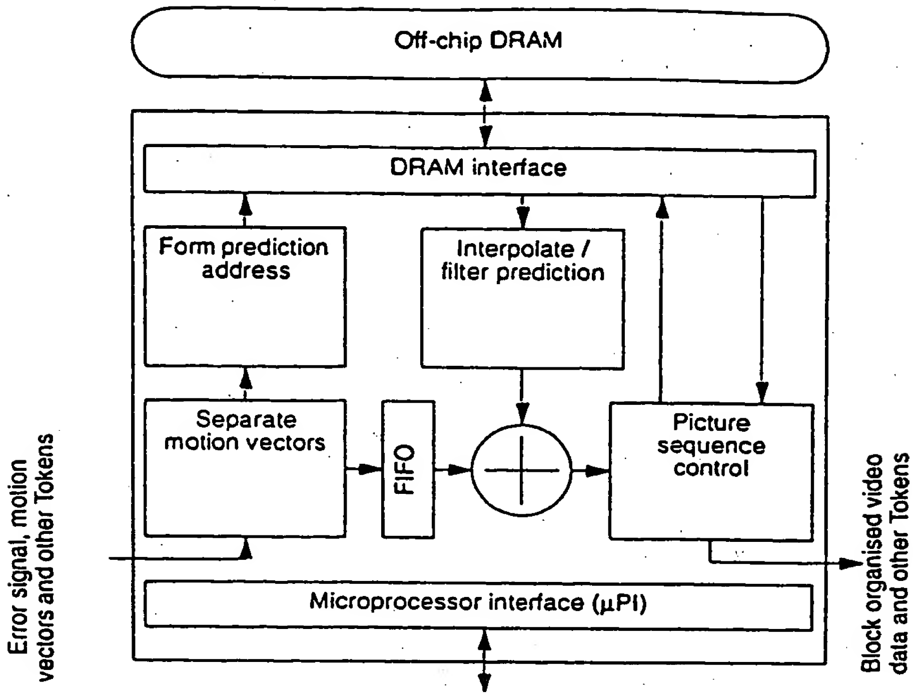

Off-chip DRAM
DRAM interface
Form prediction address
Interpolate / filter prediction
Separate motion vectors
FIFO
Picture sequence control
Microprocessor interface (µPI)
Error signal, motion vectors and other Tokens
Block organised video data and other Tokens

Figure 83

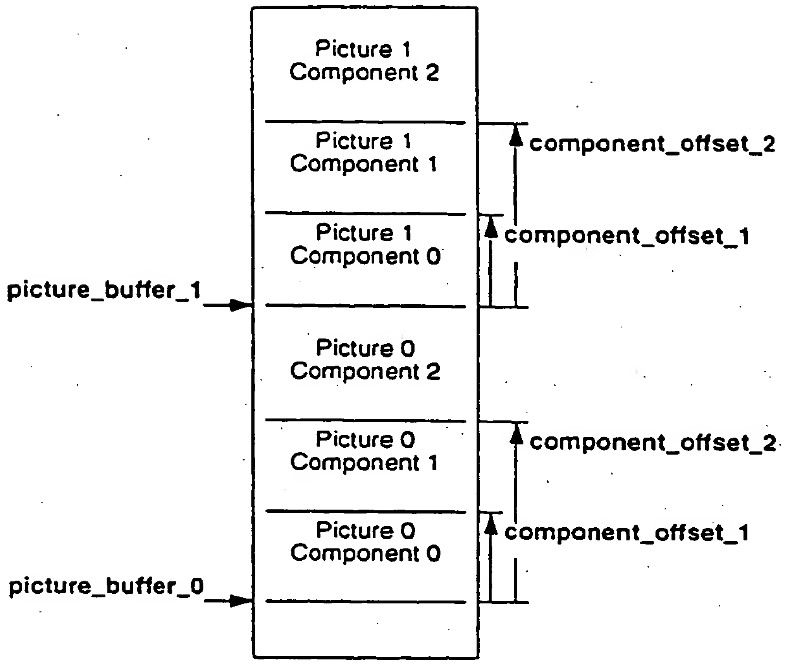


Figure 84

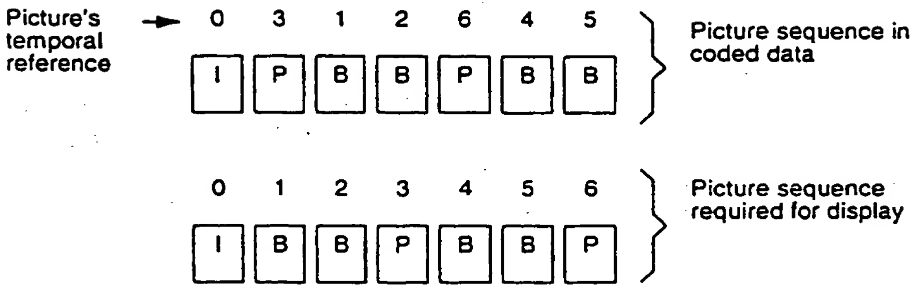

Picture's temporal reference
0 3 1 2 6 4 5
I P B B P B B
Picture sequence in coded data
0 1 2 3 4 5 6
I B B P B B P
Picture sequence required for display

Figure 85

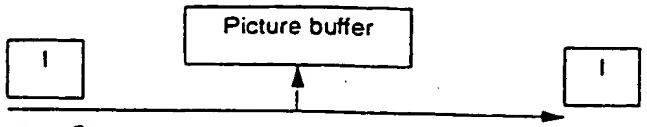


Figure 86

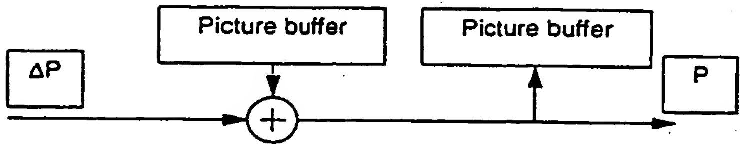


Figure 87

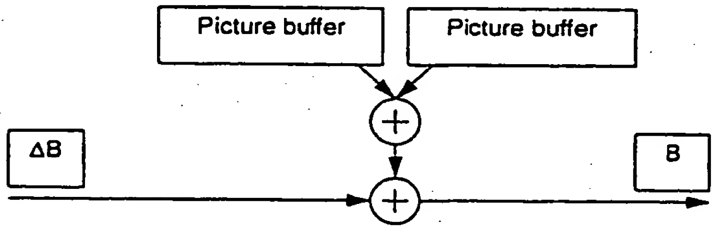


Figure. 88

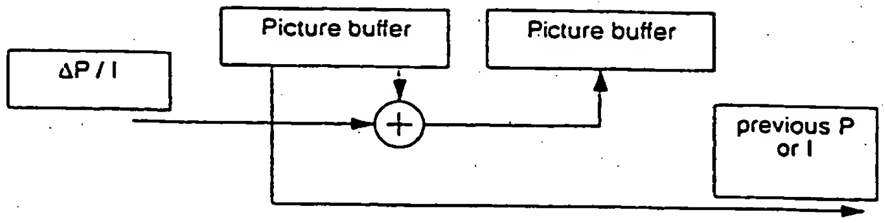


Figure 89

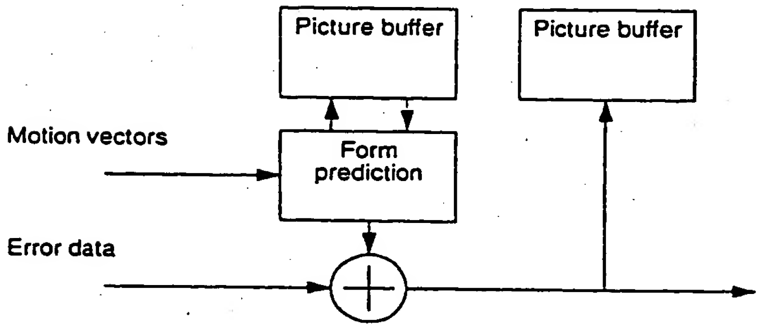


Figure 90

6 r

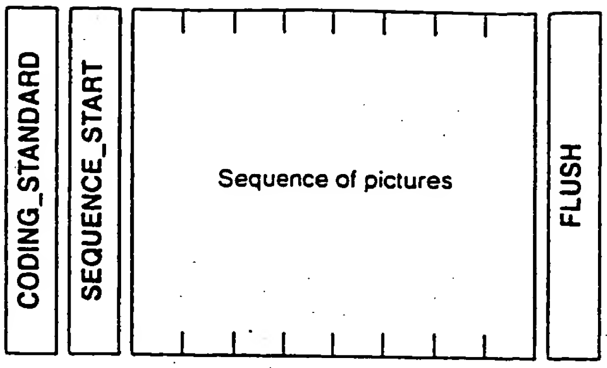


Figure 91

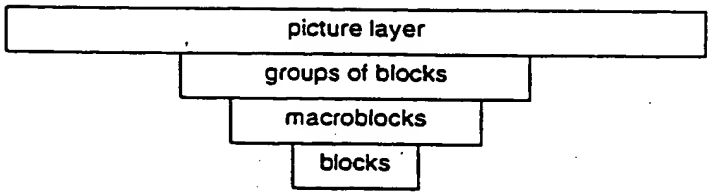


Figure 92

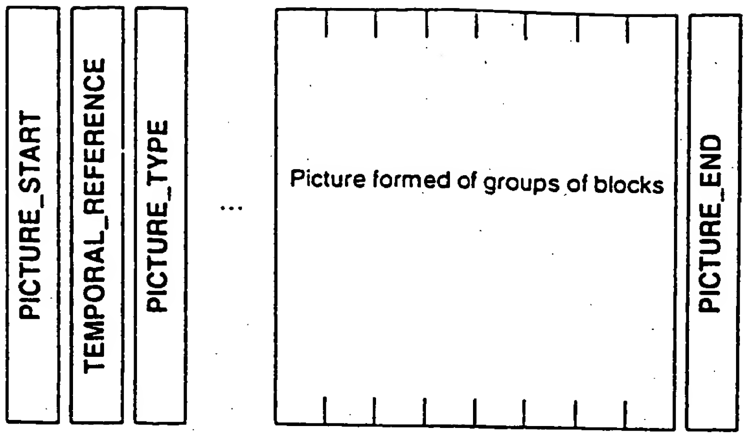


Figure 93

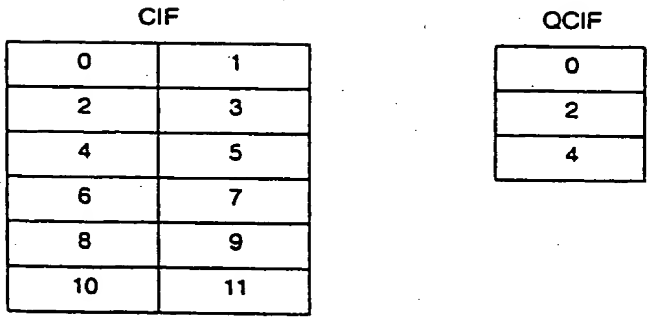


Figure 94

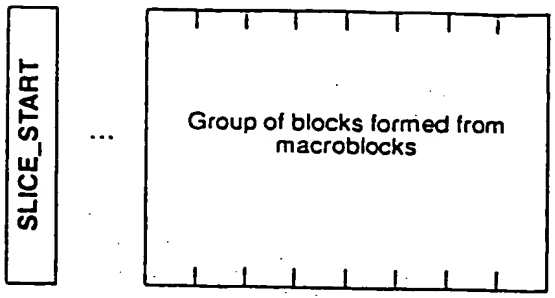


Figure 95

1	2	3	4	5	6	7	8	9	10	11
12	13	14	15	16	17	18	19	20	21	22
23	24	25	26	27	28	29	30	31	32	33

Figure 96

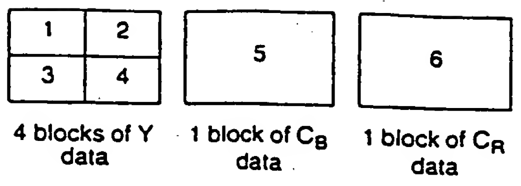


Figure 97

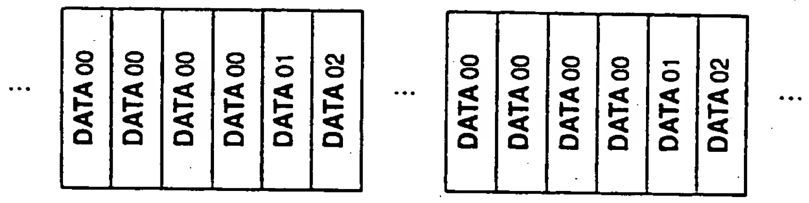


Figure 98

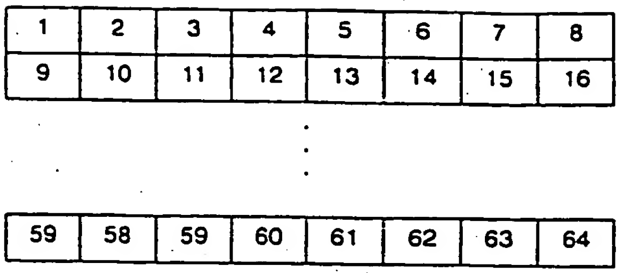


Figure 99

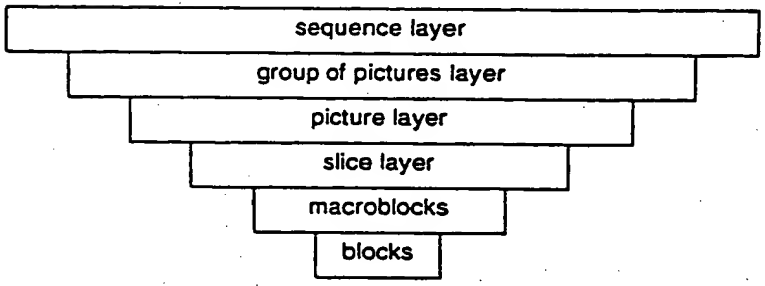


Figure 100

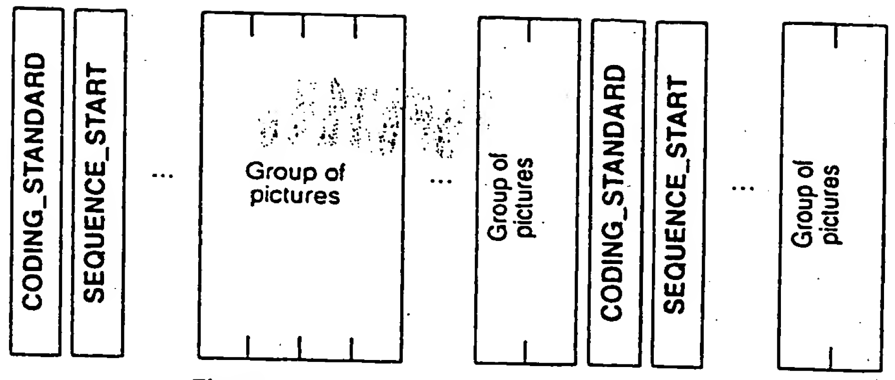


Figure 101

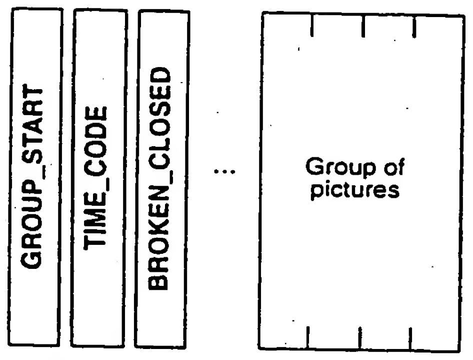


Figure 102

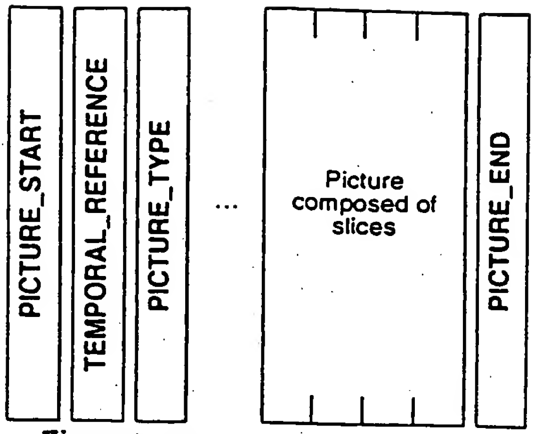


Figure 103

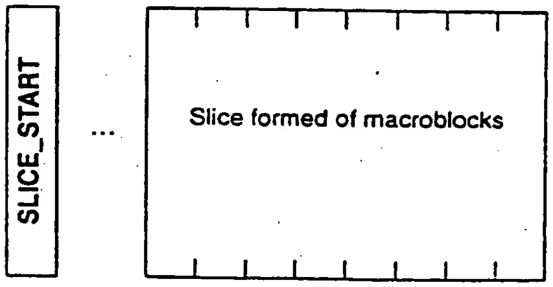


Figure 104

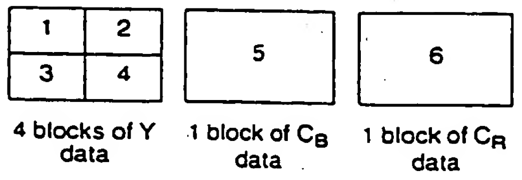


Figure. 105

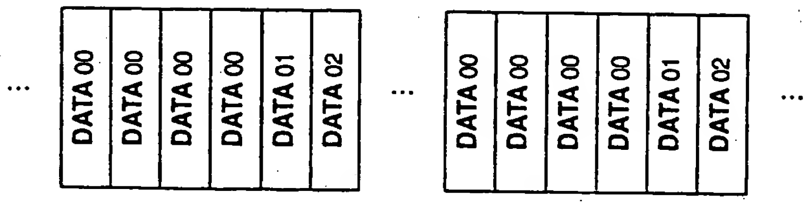


Figure 106

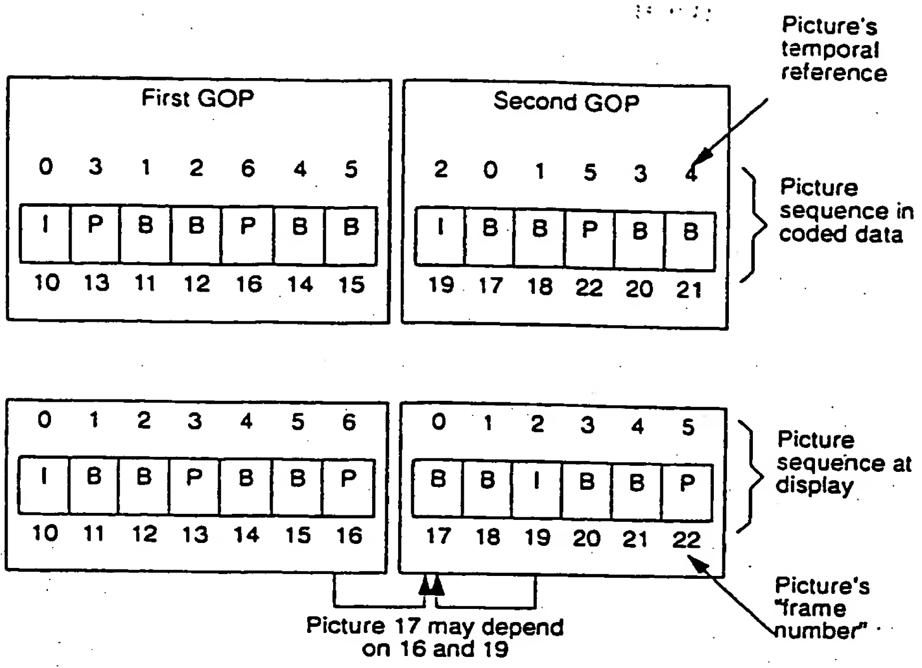

First GOP
Second GOP
0 3 1 2 6 4 5
I P B B P B B
10 13 11 12 16 14 15
2 0 1 5 3 4
I B B P B B
19 17 18 22 20 21
Picture's temporal reference
Picture sequence in coded data
0 1 2 3 4 5 6
I B B P B B P
10 11 12 13 14 15 16
0 1 2 3 4 5
B B I B B P
17 18 19 20 21 22
Picture sequence at display
Picture's "frame number"
Picture 17 may depend on 16 and 19

Figure 107

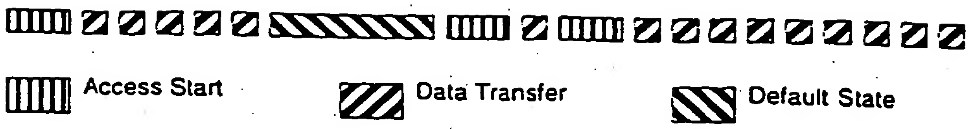


Figure 108

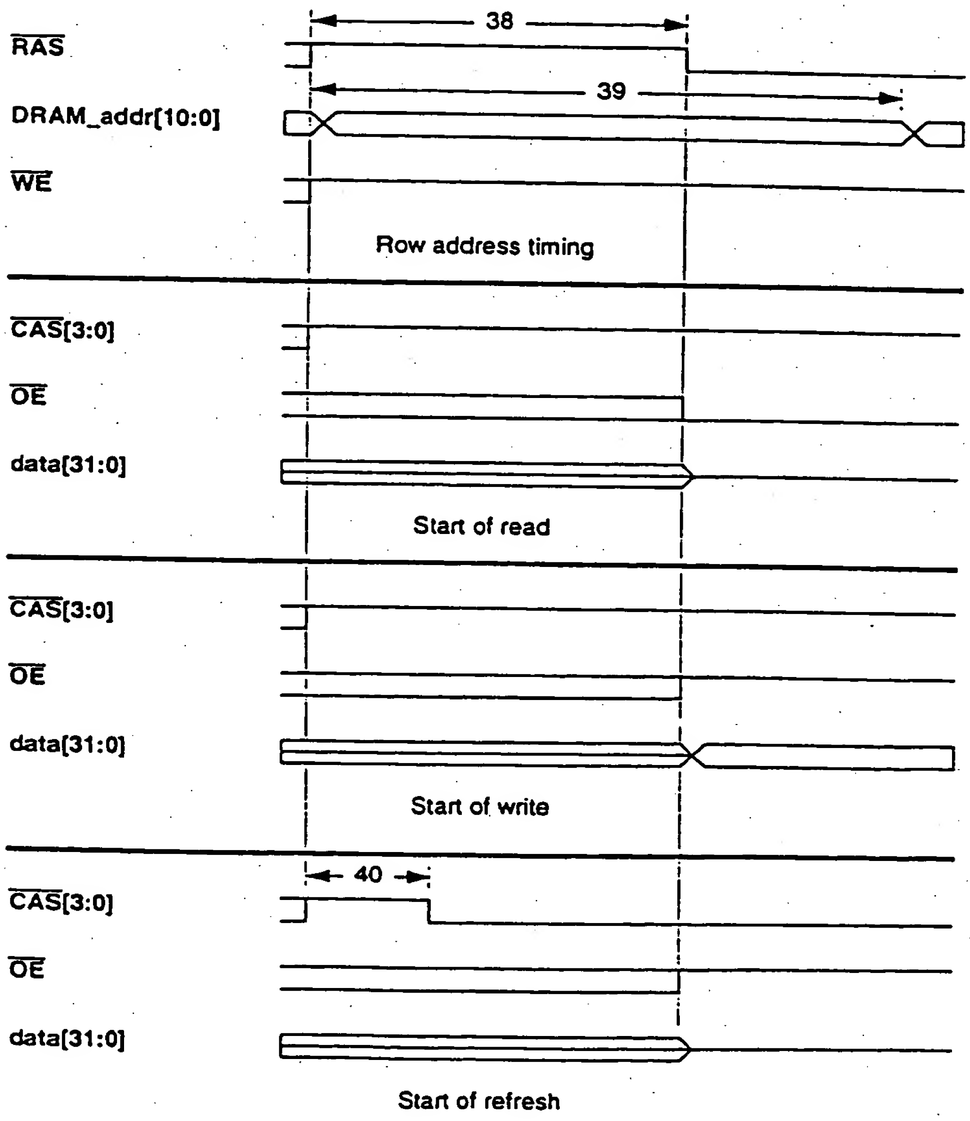


Figure 109

78

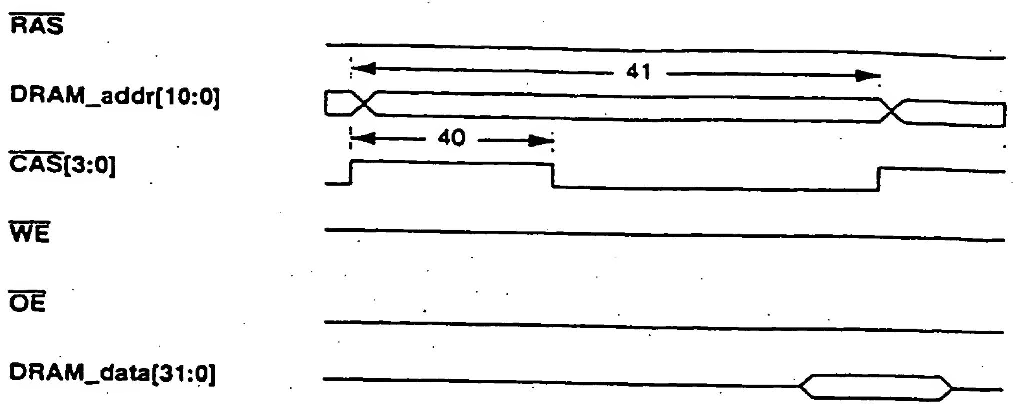


Figure . 110

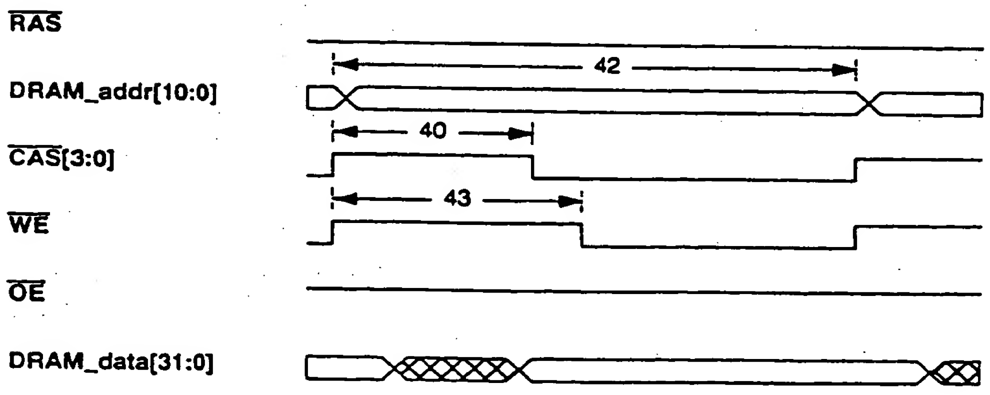


Figure . 111

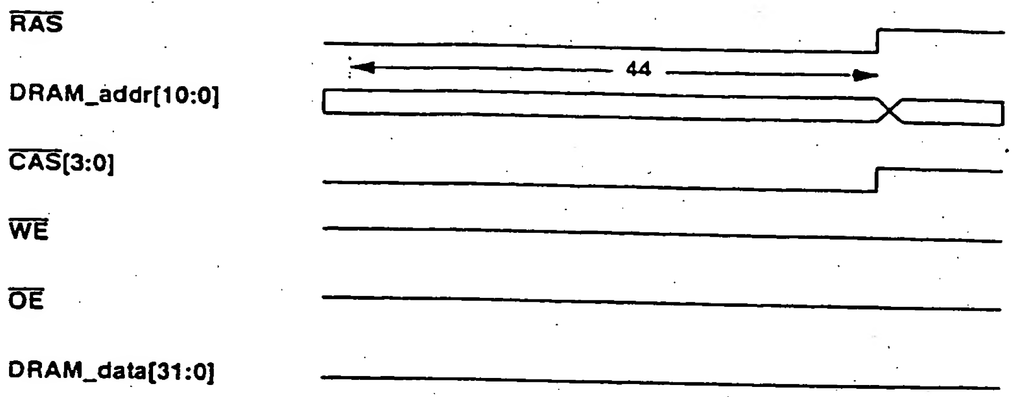


Figure 112

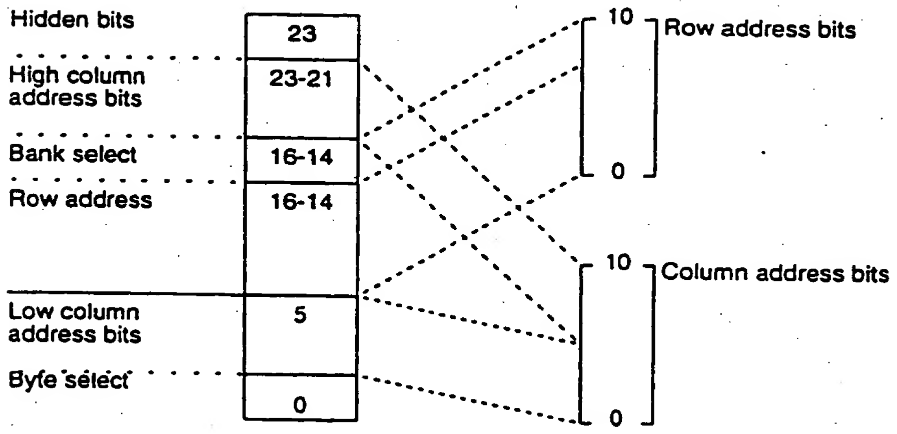


Figure. 113

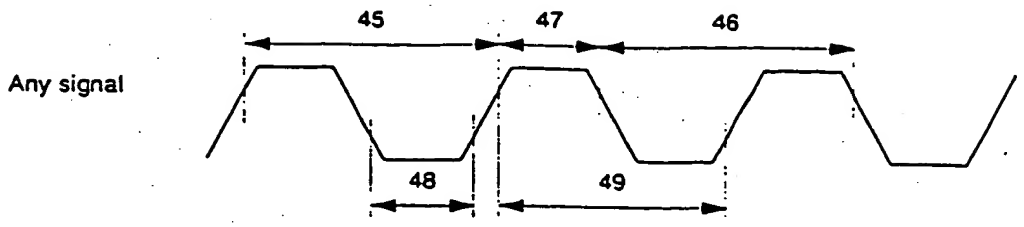


Figure 114

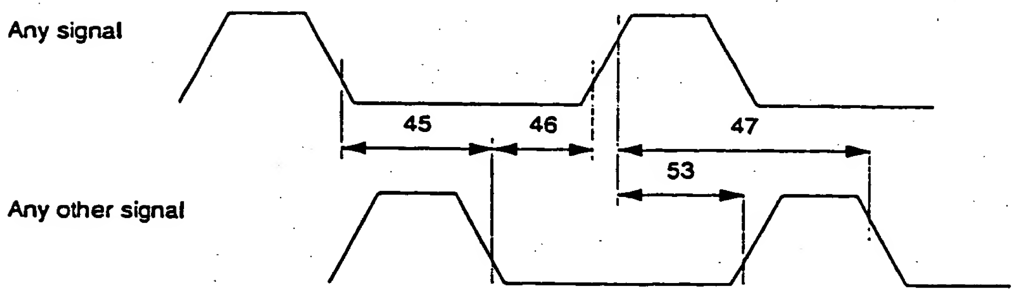


Figure 115

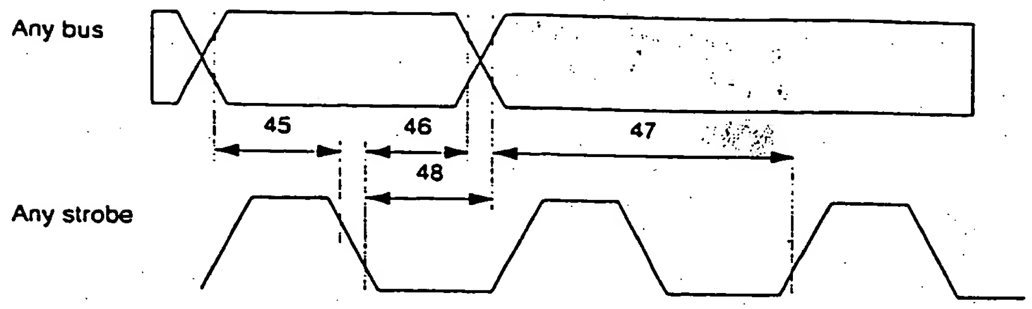


Figure 116

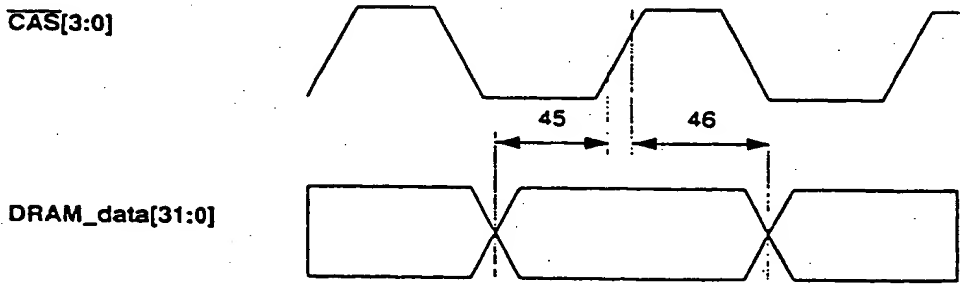


Figure 117

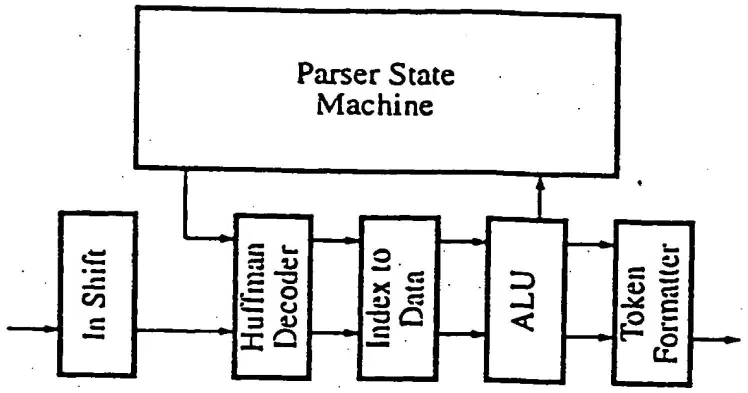

Parser State Machine
In Shift
Huffman Decoder
Index to Data
ALU
Token Formatter

Figure 118

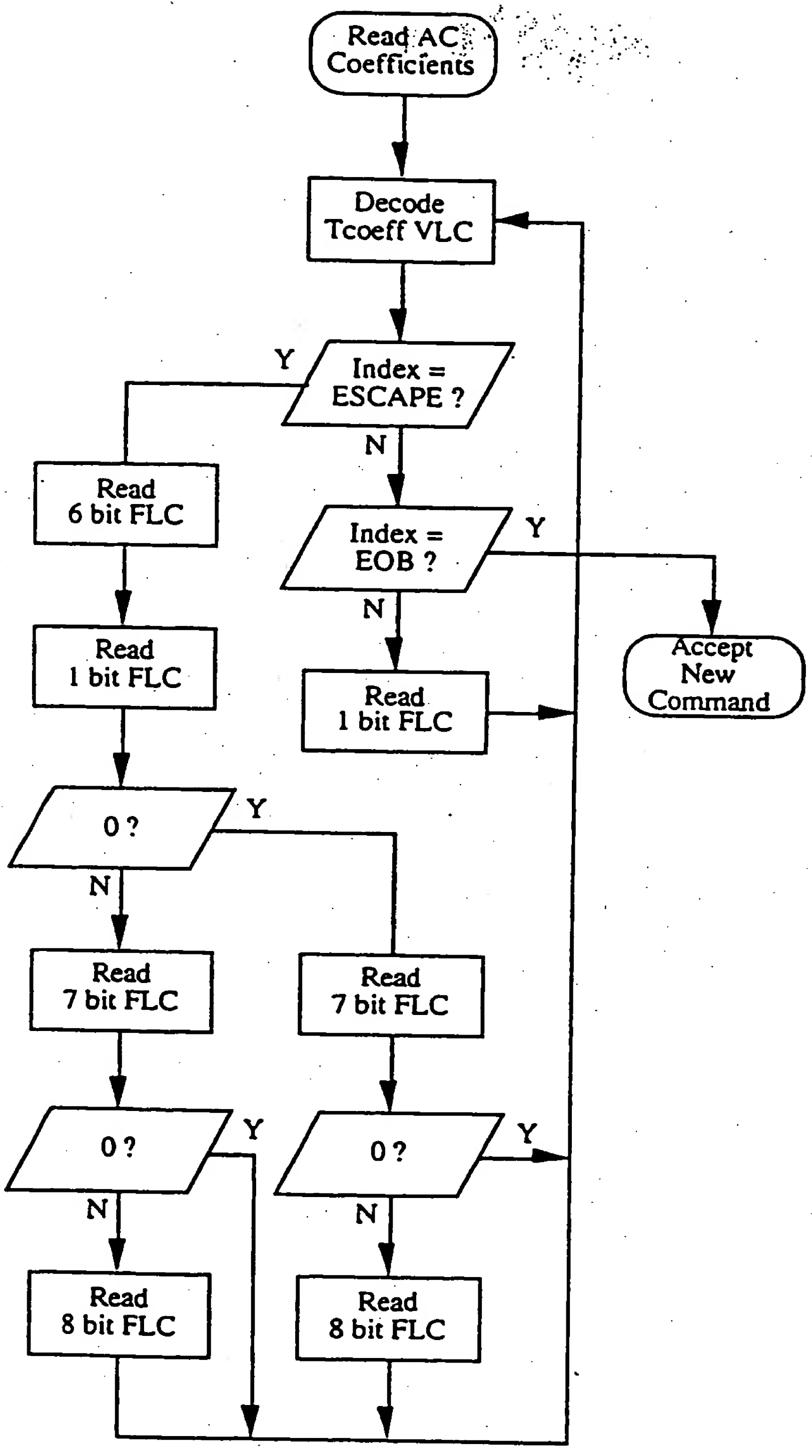
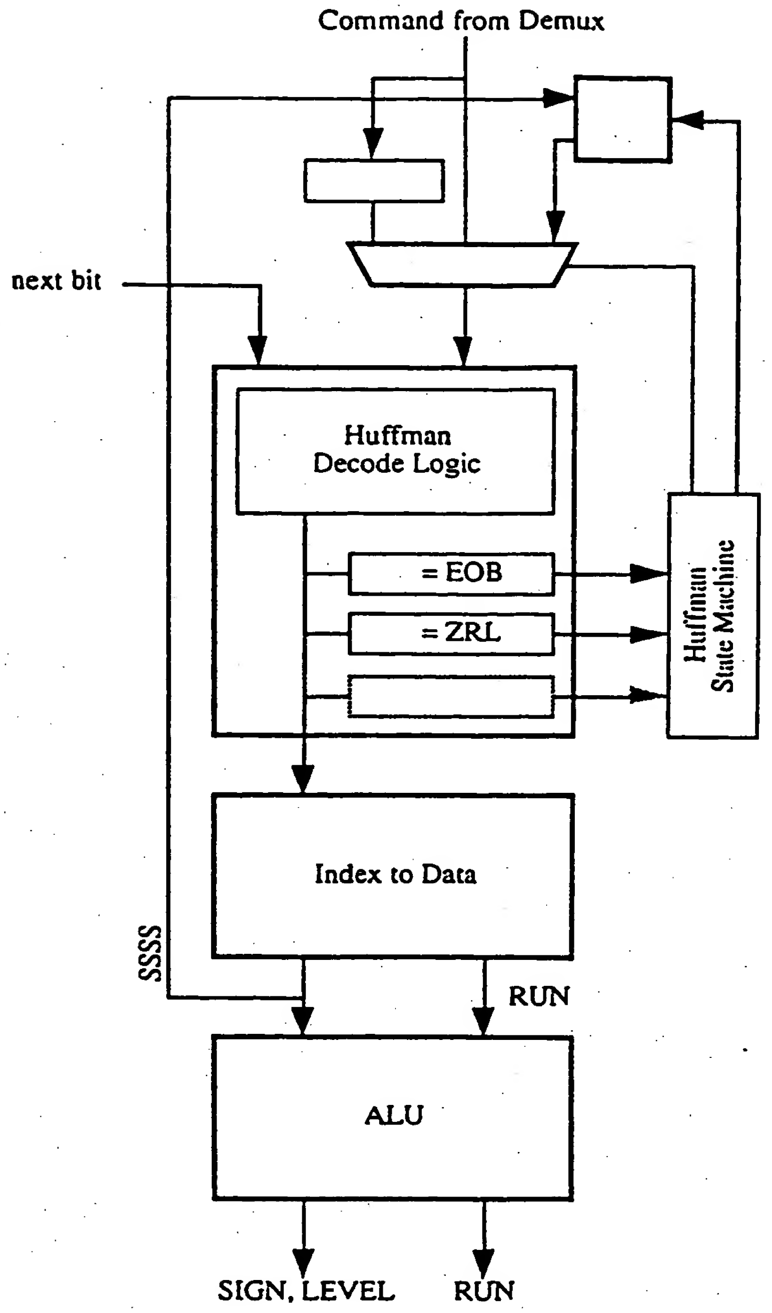

Read AC Coefficients
Decode Tcoeff VLC
Index = ESCAPE ?
Y
N
Read 6 bit FLC
Index = EOB ?
Y
N
Accept New Command
Read 1 bit FLC
Read 1 bit FLC
0 ?
Y
N
Read 7 bit FLC
Read 7 bit FLC
0 ?
Y
N
0 ?
Y
N
Read 8 bit FLC
Read 8 bit FLC

Figure 119



Command from Demux
next bit
Huffman Decode Logic
= EOB
= ZRL
Huffman State Machine
Index to Data
SSSS
RUN
ALU
SIGN, LEVEL
RUN

Figure 120

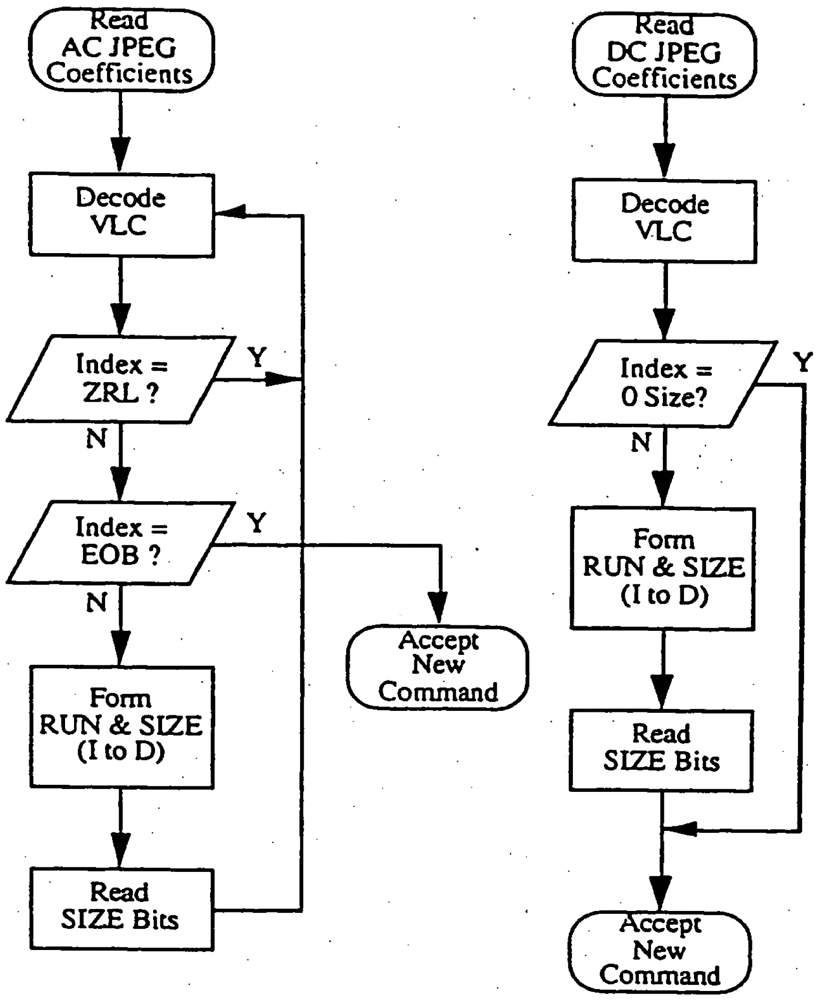

Read AC JPEG Coefficients
Decode VLC
Index = ZRL ?
Y
N
Index = EOB ?
Y
N
Form RUN & SIZE (I to D)
Read SIZE Bits
Accept New Command
Read DC JPEG Coefficients
Decode VLC
Index = 0 Size?
Y
N
Form RUN & SIZE (I to D)
Read SIZE Bits
Accept New Command

Figure 121

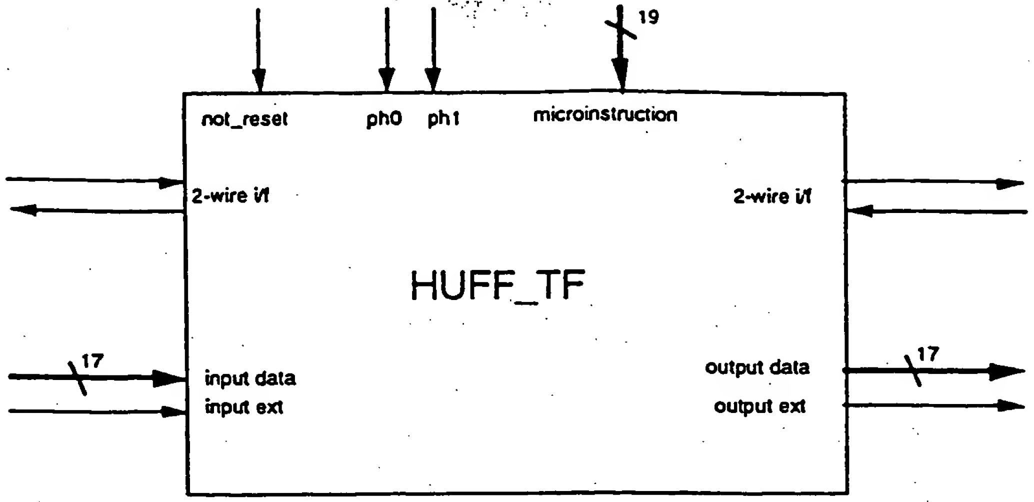


Figure 122

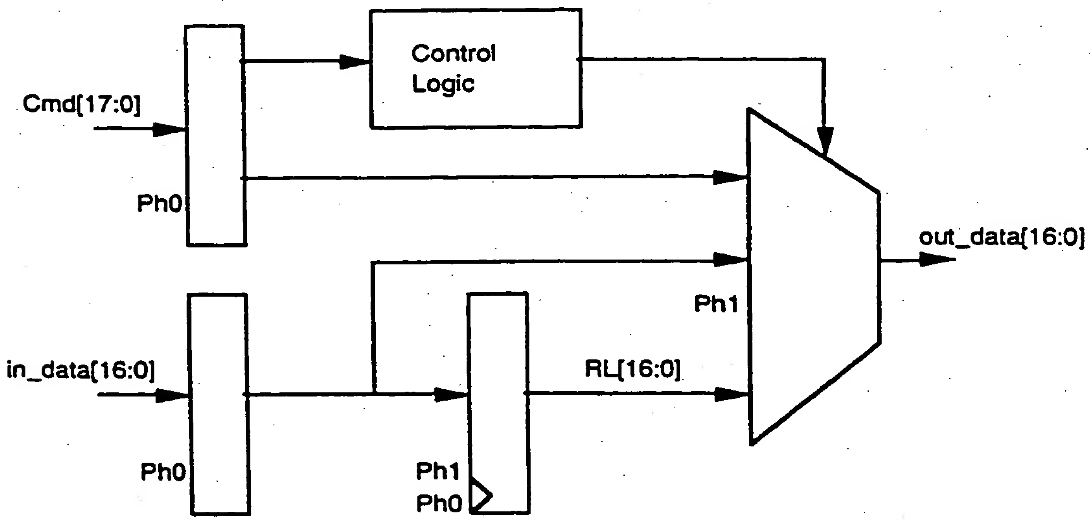


Figure 123

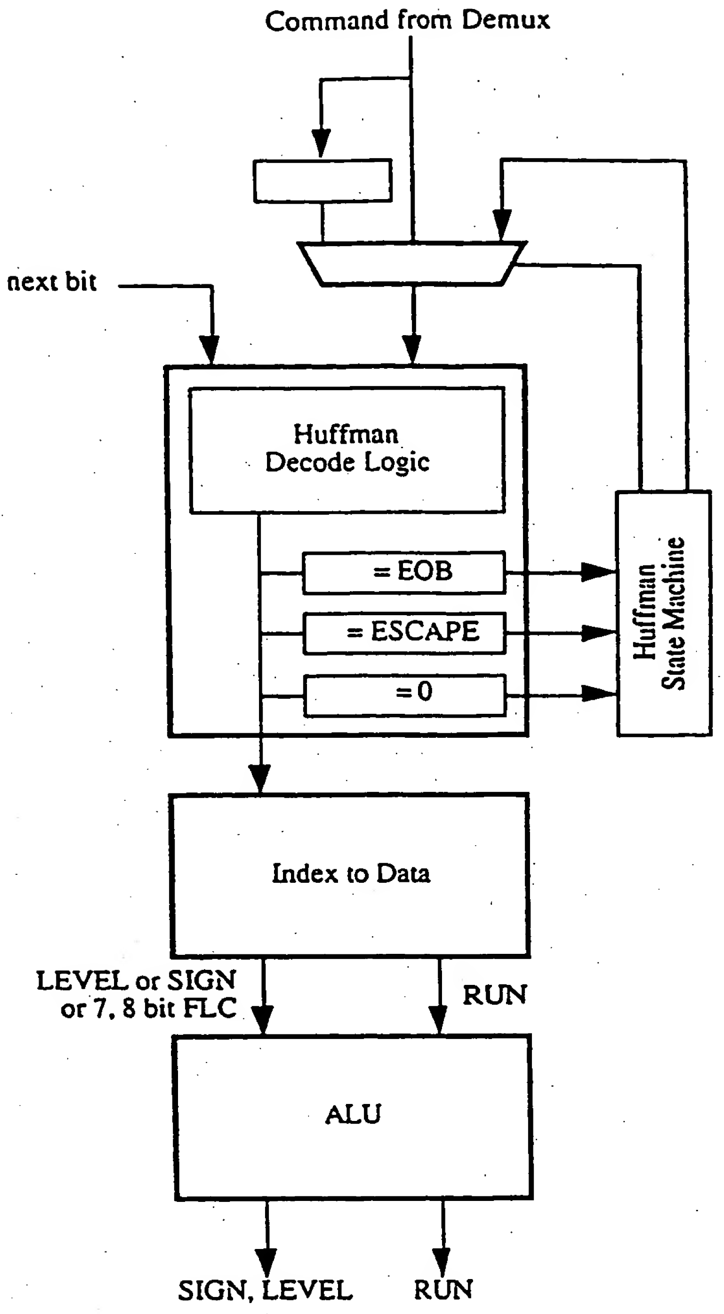


Figure 124

88

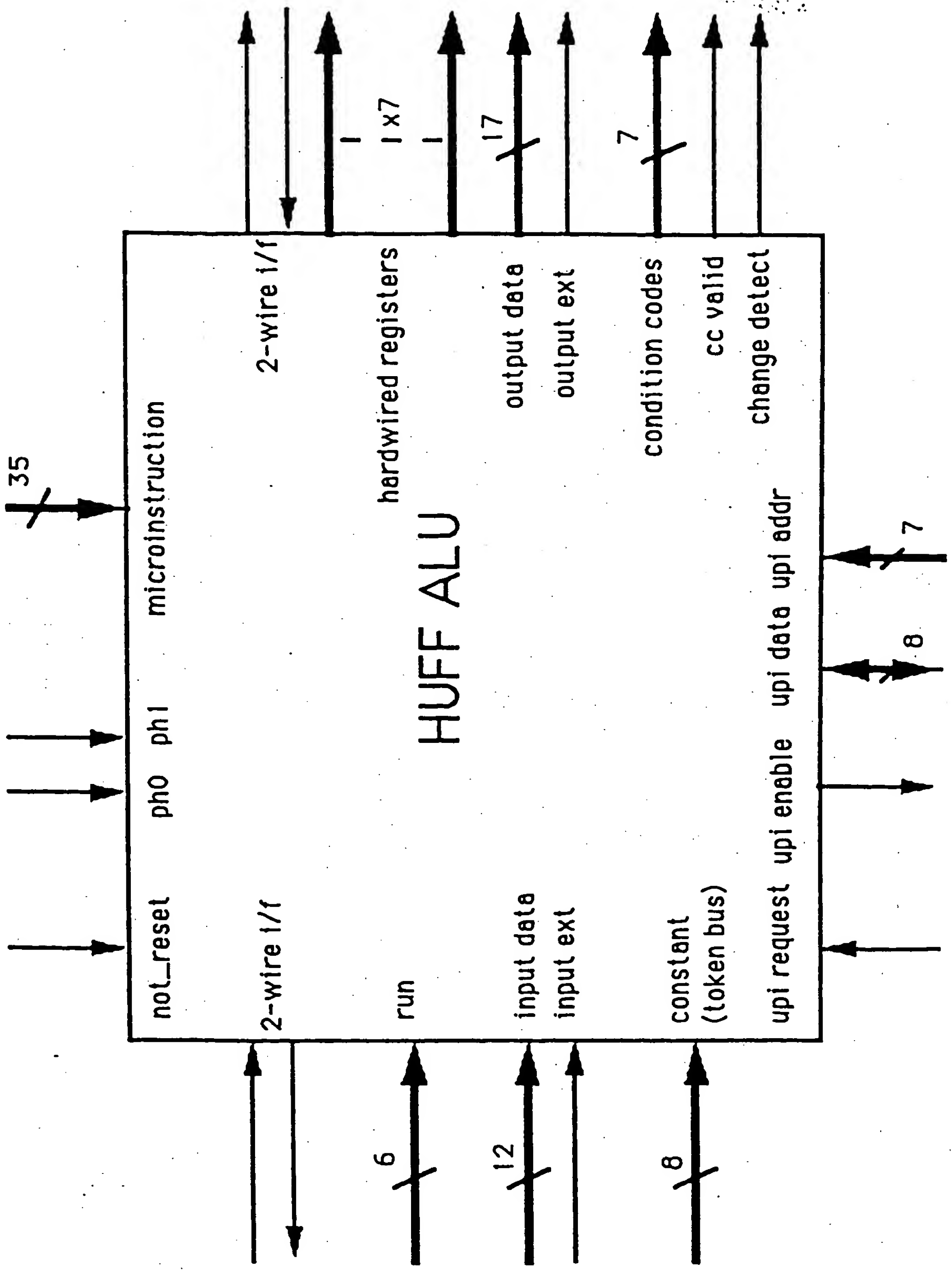


Figure 125

89

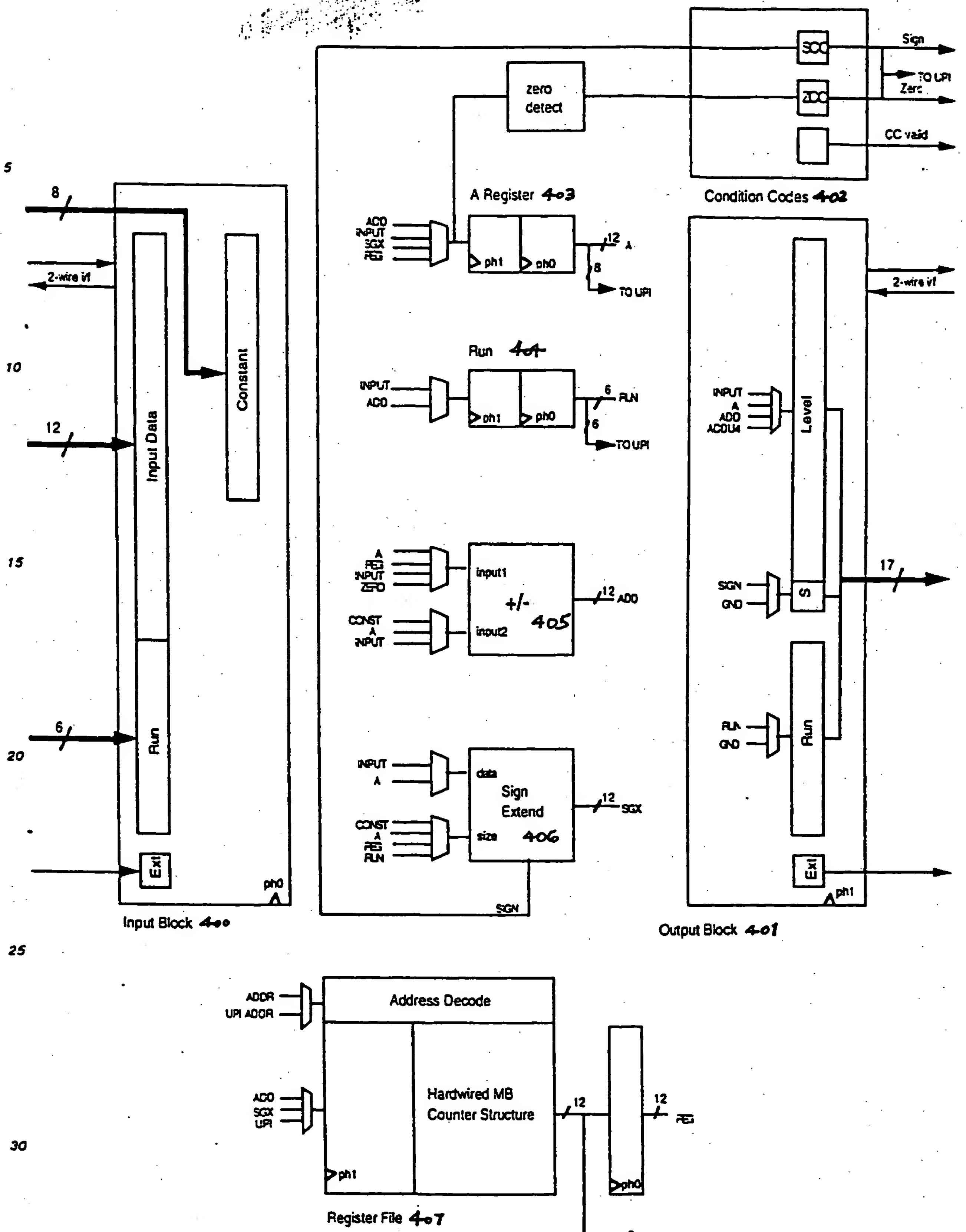

Input Block 400
Input Data
Run
Ext
Constant
2-wire i/f
8
12
6
ph0
zero detect
Condition Codes 402
SCC
ZCC
Sign
TO UPI
Zero
CC valid
A Register 403
ADD
INPUT
SGX
REG
ph1
ph0
12
A
8
TO UPI
Run 404
INPUT
ADD
6
RUN
6
TO UPI
A
REG
INPUT
ZERO
input1
+/-
405
CONST
A
INPUT
input2
12
ADD
INPUT
A
data
Sign Extend
CONST
A
REG
RUN
size
406
12
SGX
SIGN
Output Block 401
Level
INPUT
A
ADD
ACOU4
2-wire i/f
SIGN
GND
S
17
RUN
GND
Run
Ext
ph1
Address Decode
ADDR
UPI ADDR
ADD
SGX
UPI
Hardwired MB Counter Structure
ph1
12
12
REG
ph0
8
TO UPI
Register File 407

Figure 126

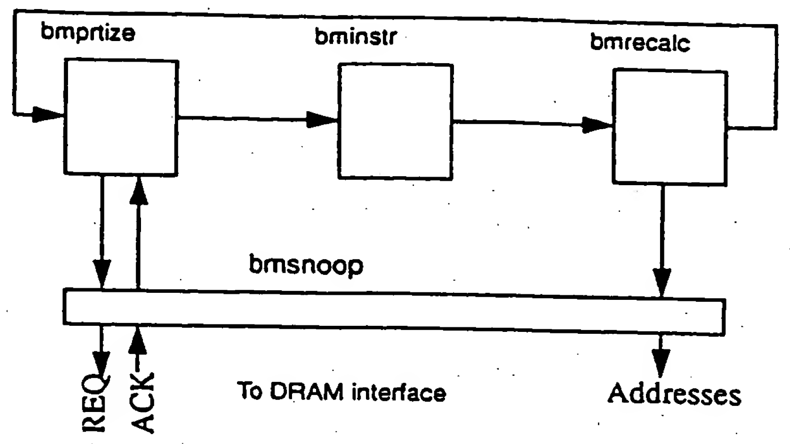

bmprtize
bminstr
bmrecalc
bmsnoop
REQ
ACK
To DRAM interface
Addresses

Figure 127

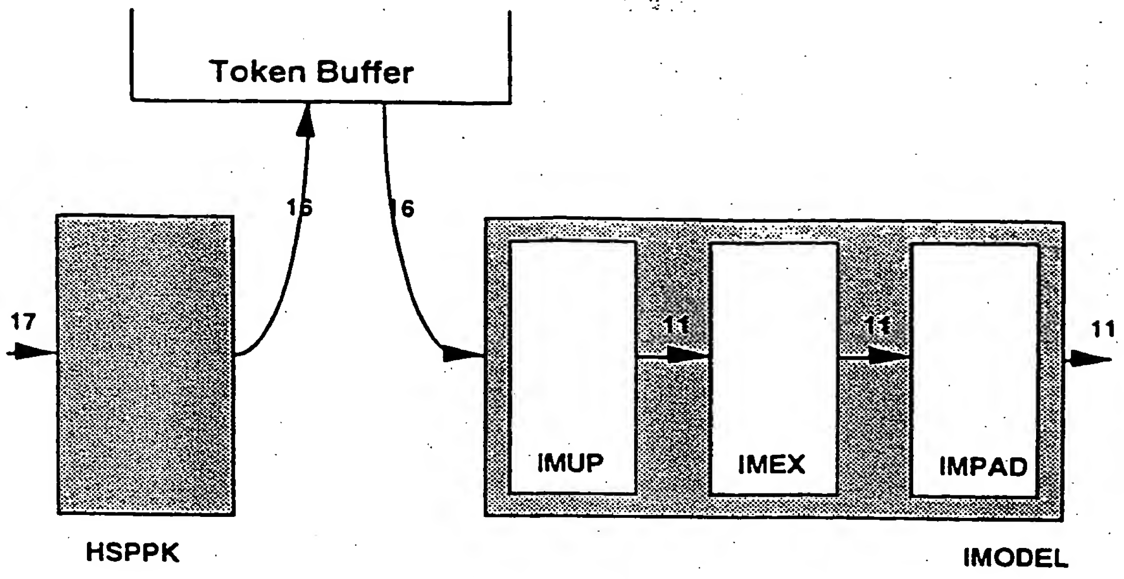


Figure 128

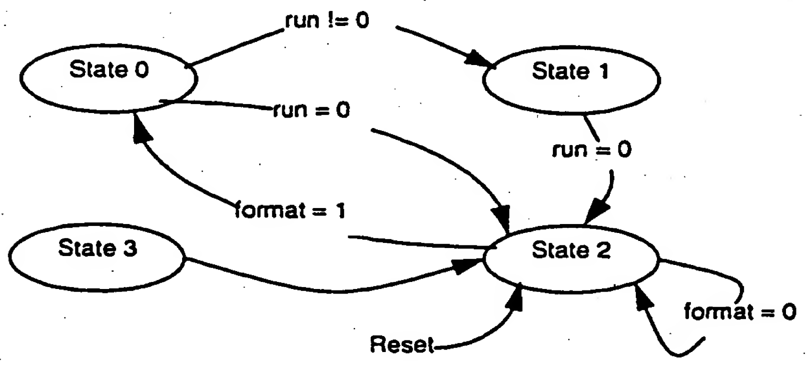


Figure 129

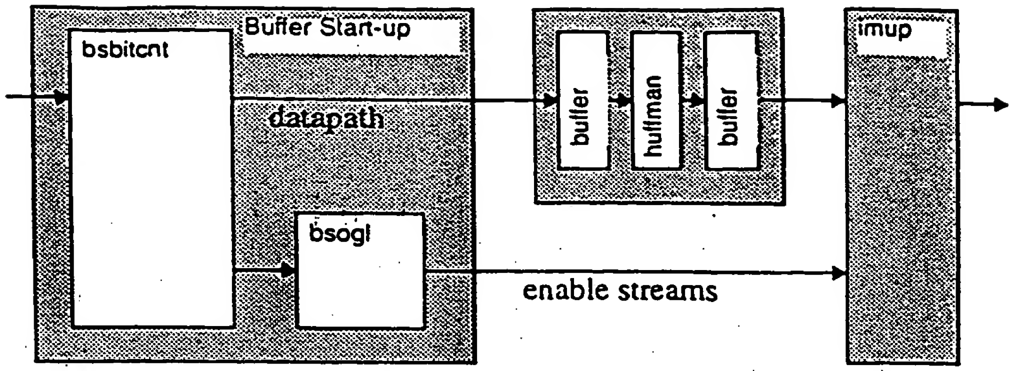

Buffer Start-up
bsbitcnt
datapath
bsogl
buffer
huffman
buffer
imup
enable streams

Figure 130

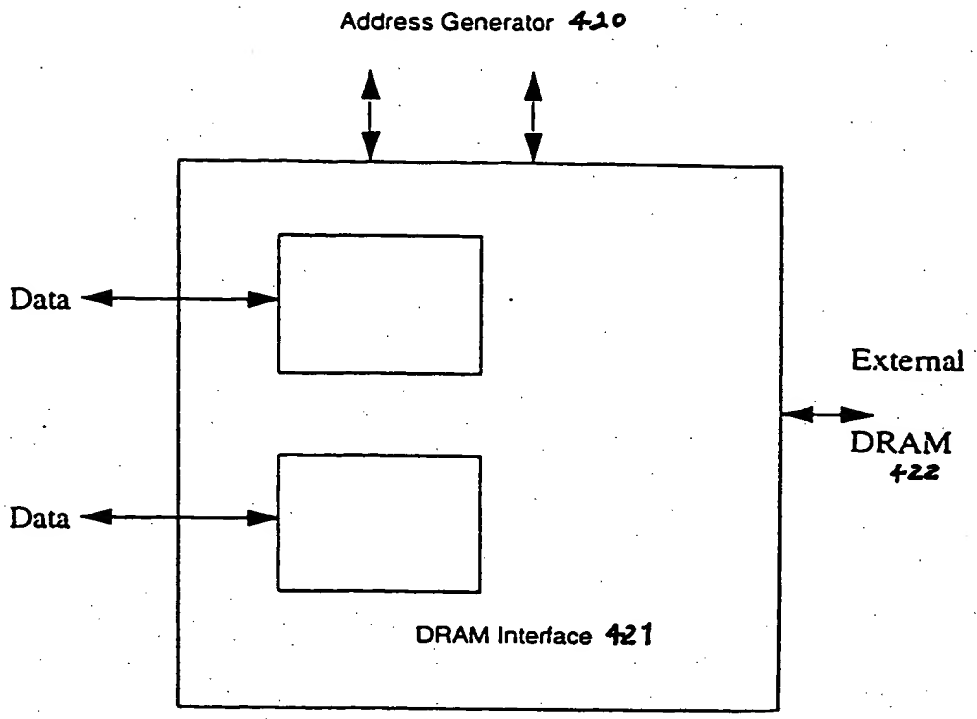

Address Generator 410
Data
Data
External
DRAM
422
DRAM Interface 421

Figure 131

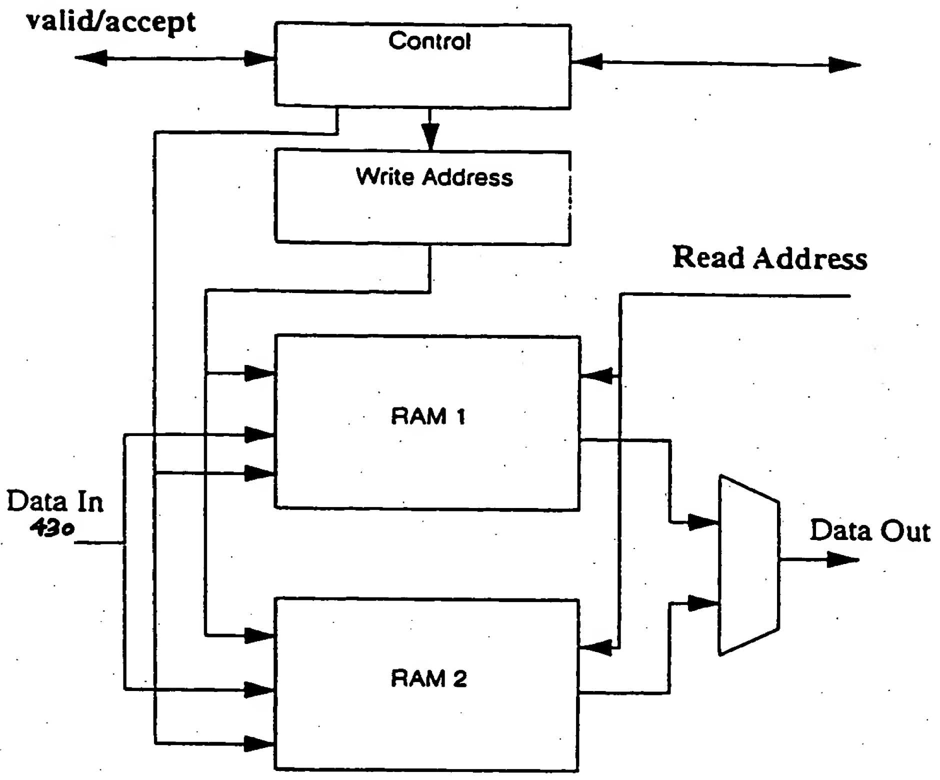

valid/accept
Control
Write Address
Read Address
RAM 1
Data In
430
Data Out
RAM 2

Figure 132

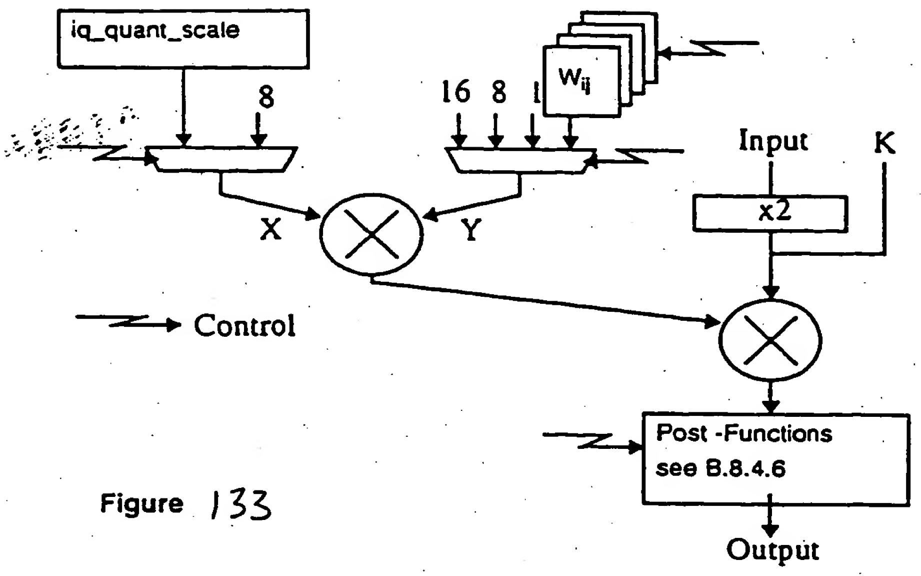


Figure 133

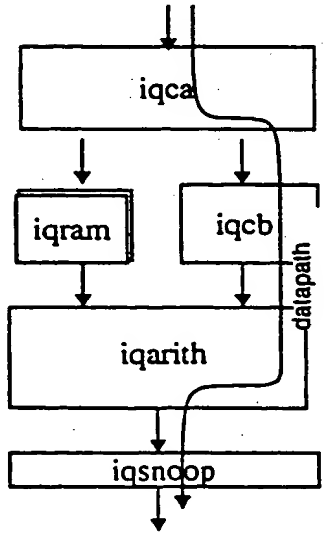


Figure 134

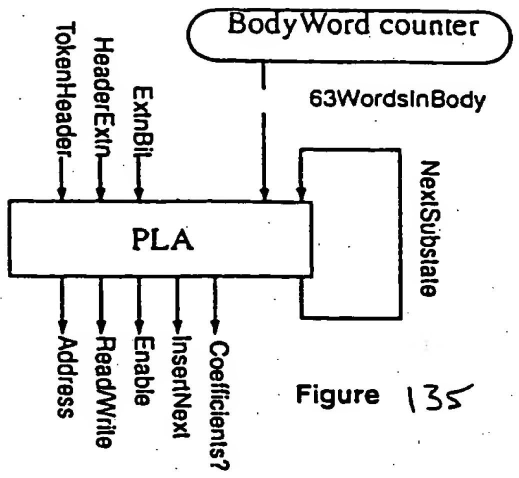


Figure 135

97

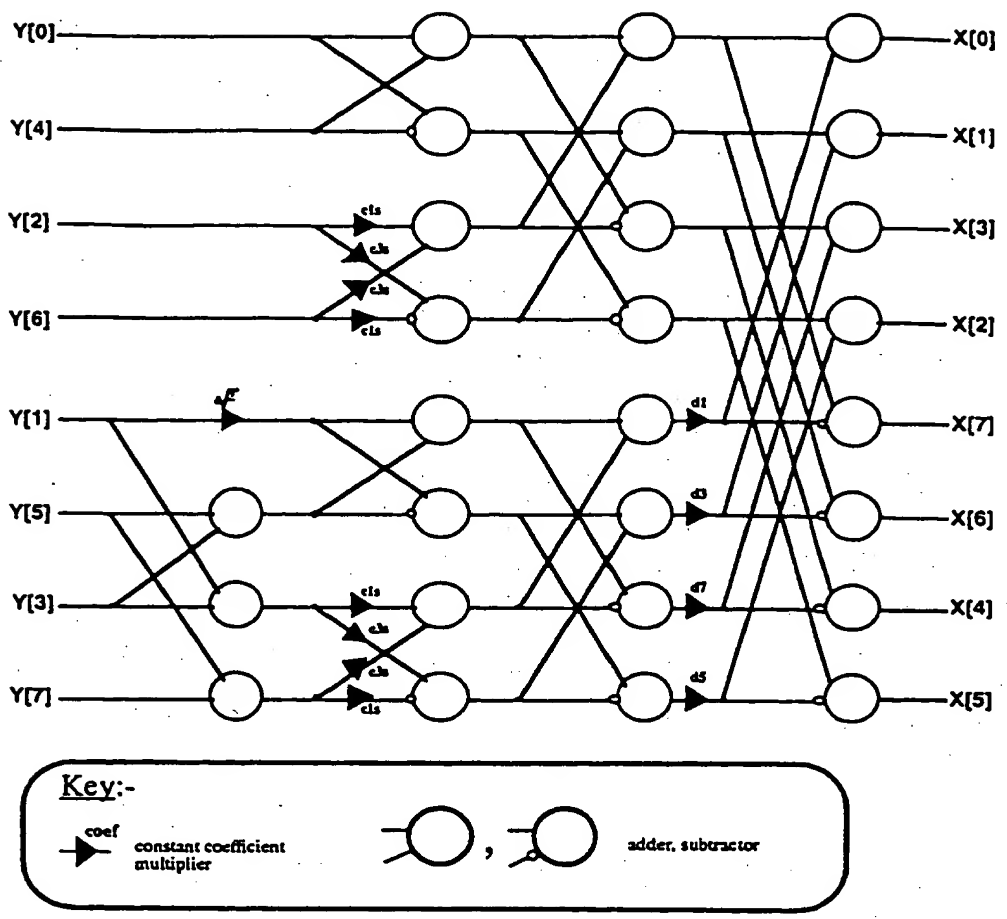

Y[0]
Y[4]
Y[2]
Y[6]
Y[1]
Y[5]
Y[3]
Y[7]
c1s
c1s
c1s
c1s
√2
c1s
c1s
c1s
c1s
d1
d3
d7
d5
X[0]
X[1]
X[3]
X[2]
X[7]
X[6]
X[4]
X[5]
Key:-
coef
constant coefficient multiplier
adder, subtractor

Figure: 136

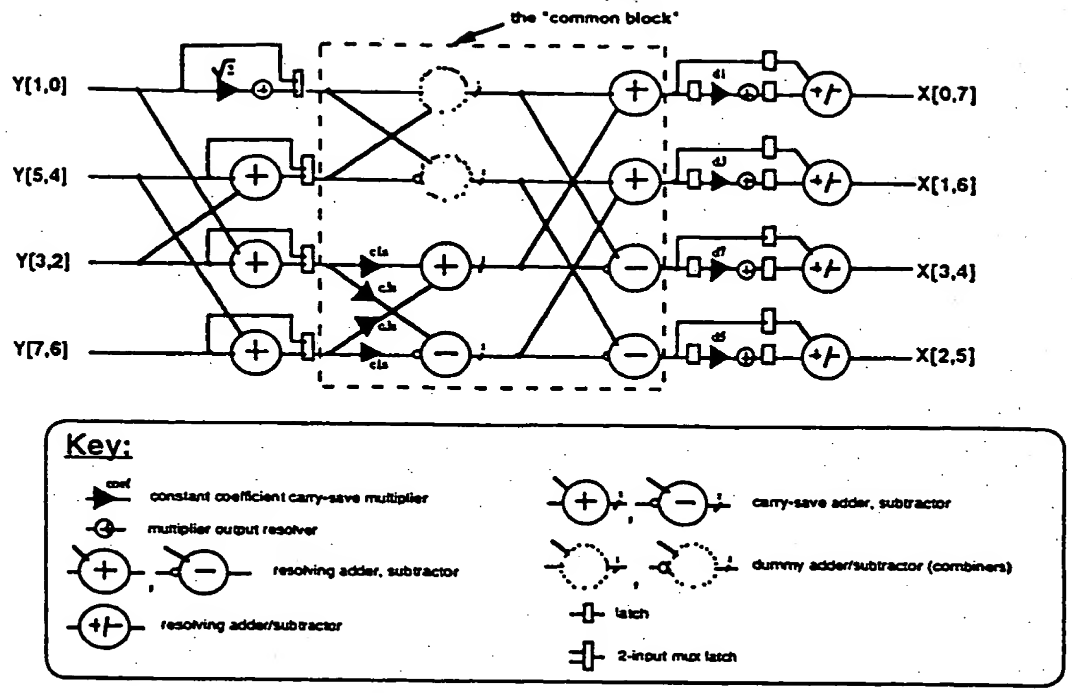

the "common block"
Y[1,0]
Y[5,4]
Y[3,2]
Y[7,6]
X[0,7]
X[1,6]
X[3,4]
X[2,5]
Key:
constant coefficient carry-save multiplier
multiplier output resolver
resolving adder, subtractor
resolving adder/subtractor
carry-save adder, subtractor
dummy adder/subtractor (combiners)
latch
2-input mux latch

Figure 137

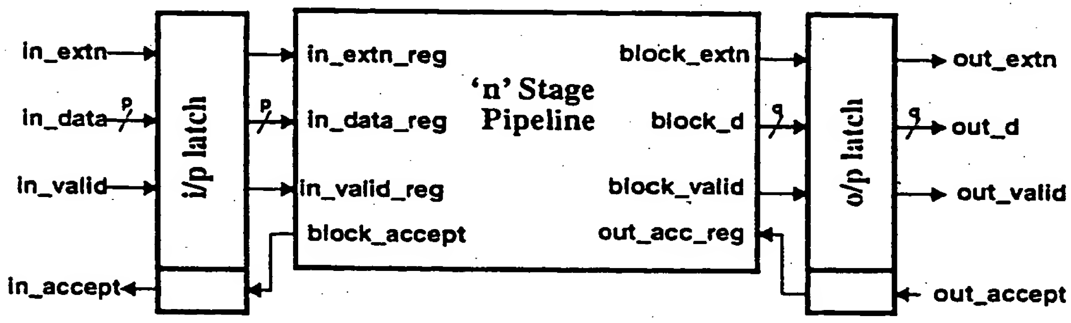


Figure 138

G9

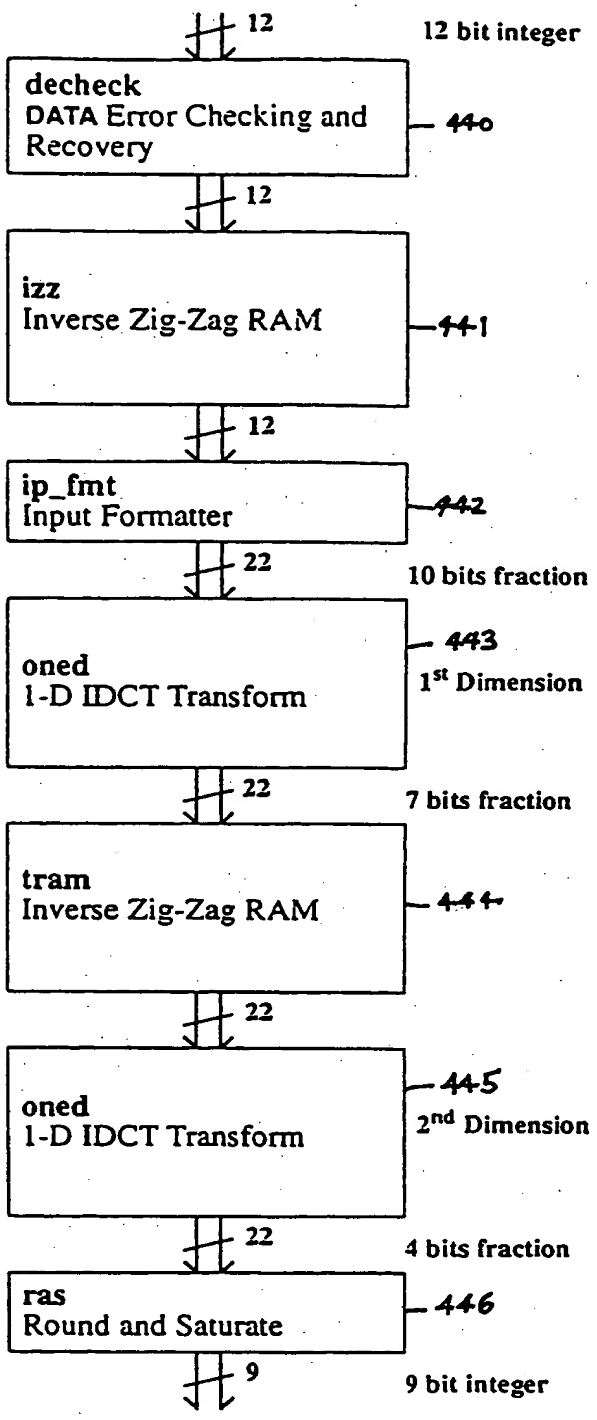

12
12 bit integer
decheck
DATA Error Checking and Recovery
440
12
izz
Inverse Zig-Zag RAM
441
12
ip_fmt
Input Formatter
442
22
10 bits fraction
oned
1-D IDCT Transform
443
1st Dimension
22
7 bits fraction
tram
Inverse Zig-Zag RAM
444
22
oned
1-D IDCT Transform
445
2nd Dimension
22
4 bits fraction
ras
Round and Saturate
446
9
9 bit integer

Figure 139

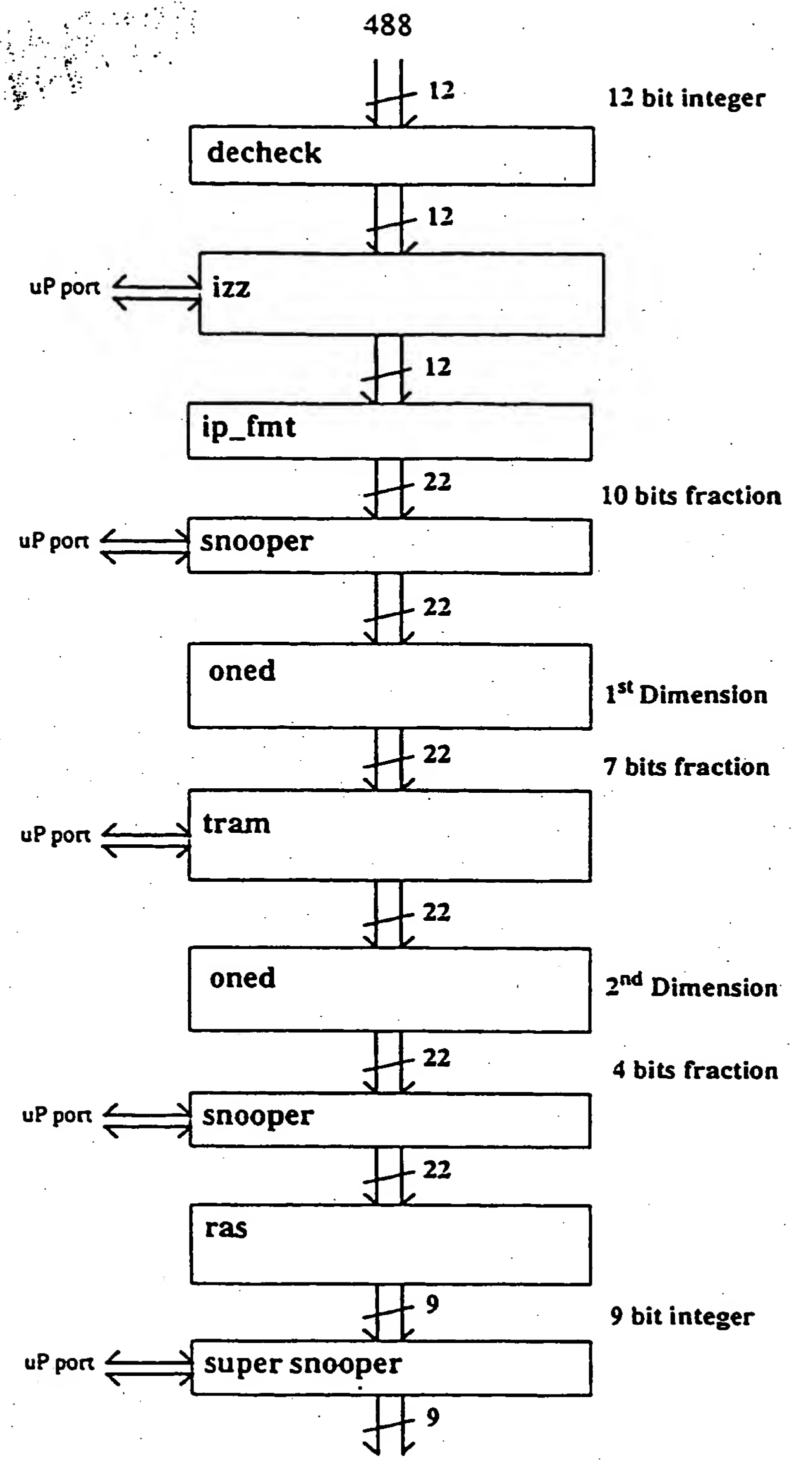
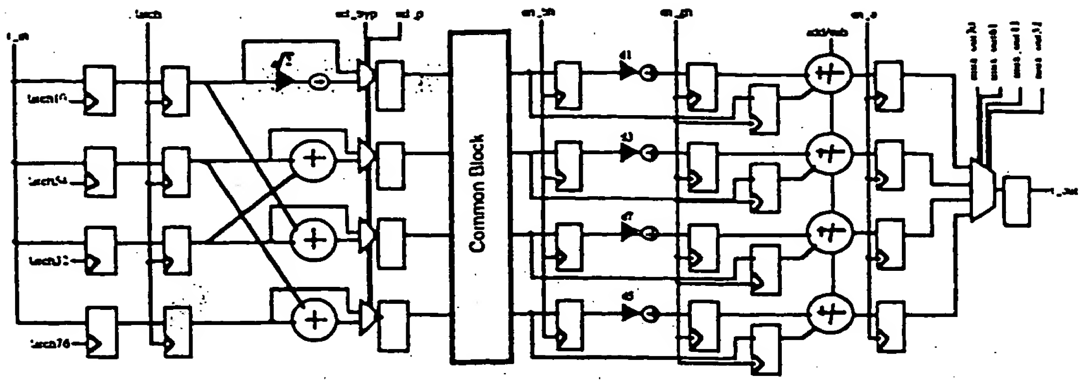
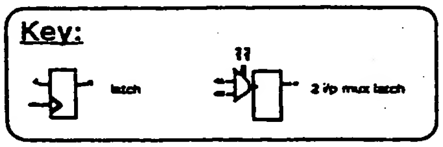

488
12
12 bit integer
decheck
12
uP port
izz
12
ip_fmt
22
10 bits fraction
uP port
snooper
22
oned
1st Dimension
22
7 bits fraction
uP port
tram
22
oned
2nd Dimension
22
4 bits fraction
uP port
snooper
22
ras
9
9 bit integer
uP port
super snooper
9

Figure 140





NOTE: "Common Block" is entirely combinatorial (no latching)

Figure 141

102

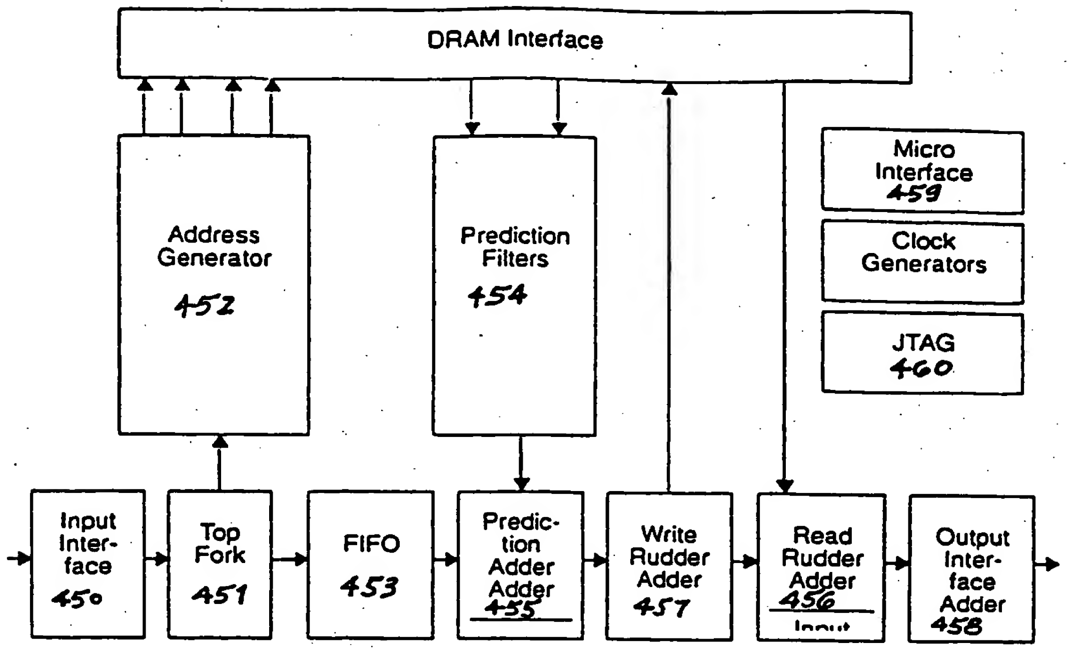

DRAM Interface
Address Generator 452
Prediction Filters 454
Micro Interface 459
Clock Generators
JTAG 460
Input Inter-face 450
Top Fork 451
FIFO 453
Predic-tion Adder Adder 455
Write Rudder Adder 457
Read Rudder Adder 456 Input
Output Inter-face Adder 458

Figure 142

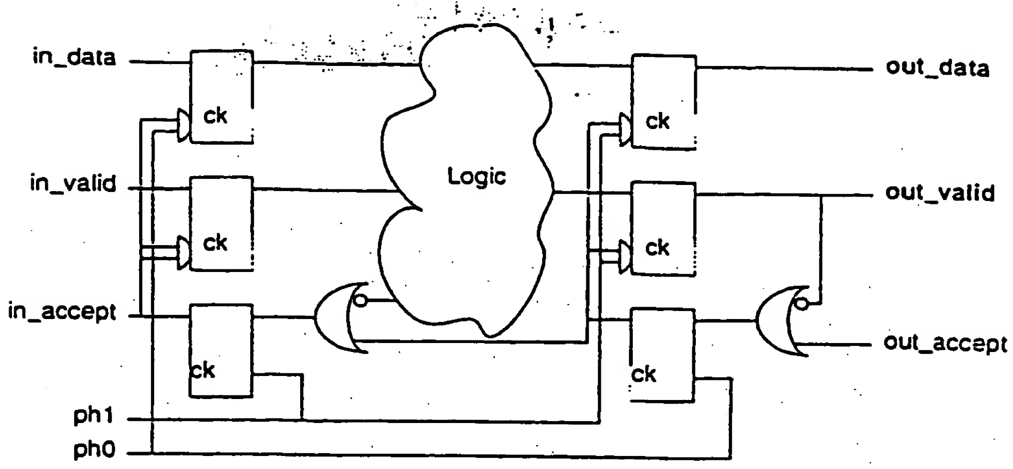
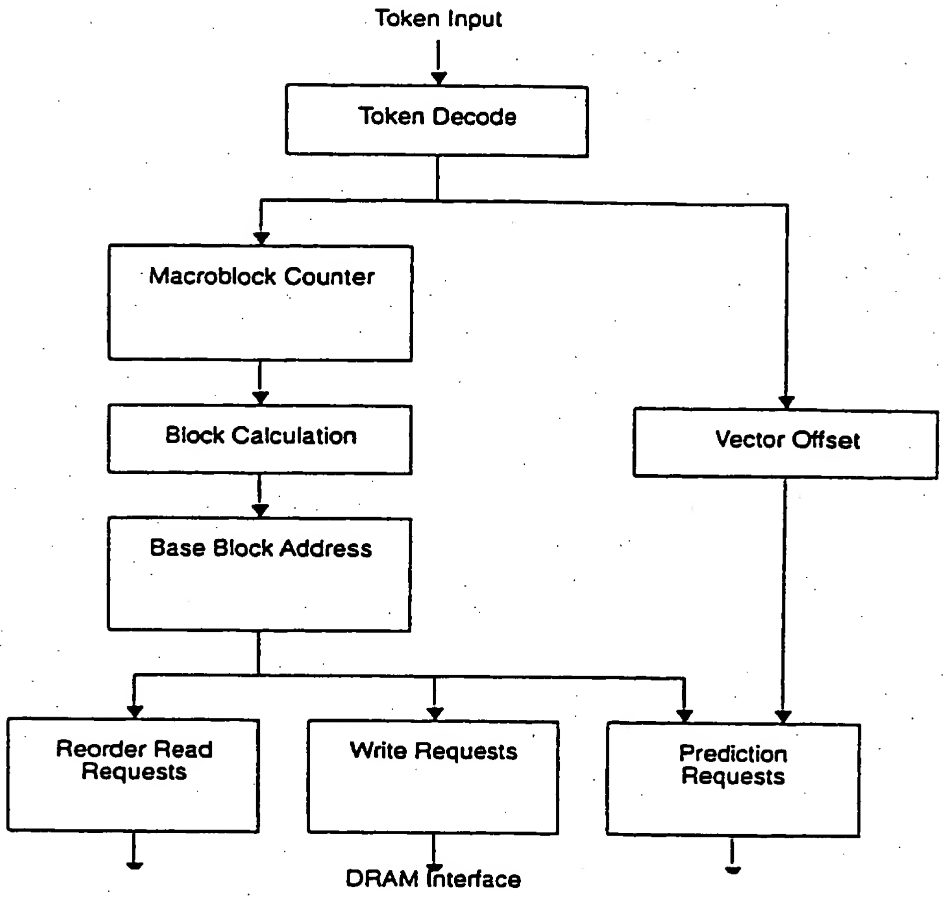

in_data
in_valid
in_accept
ph1
ph0
ck
ck
ck
Logic
ck
ck
ck
out_data
out_valid
out_accept

Figure 143



Token Input
Token Decode
Macroblock Counter
Block Calculation
Base Block Address
Vector Offset
Reorder Read Requests
Write Requests
Prediction Requests
DRAM Interface

Figure 144

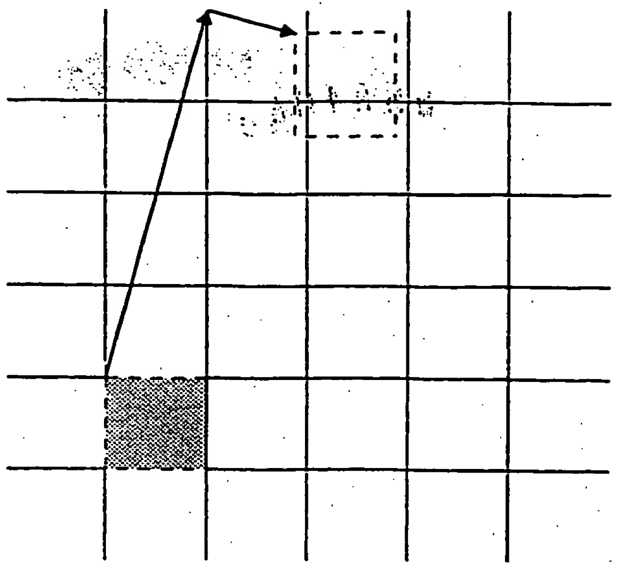


Figure 145

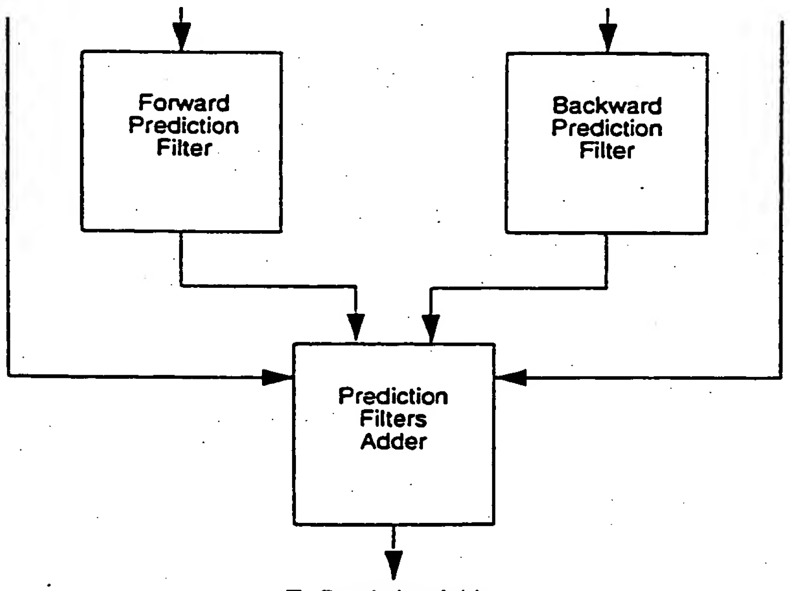


Figure 146

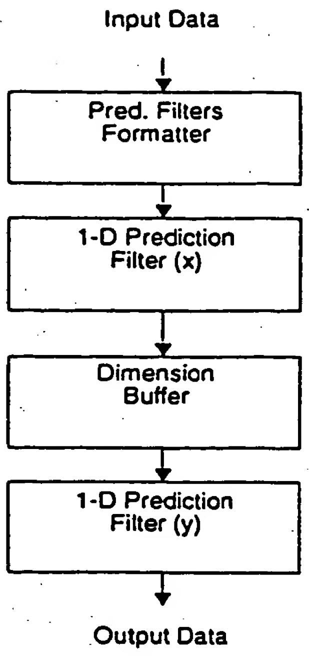

Input Data
Pred. Filters Formatter
1-D Prediction Filter (x)
Dimension Buffer
1-D Prediction Filter (y)
Output Data

Figure 147

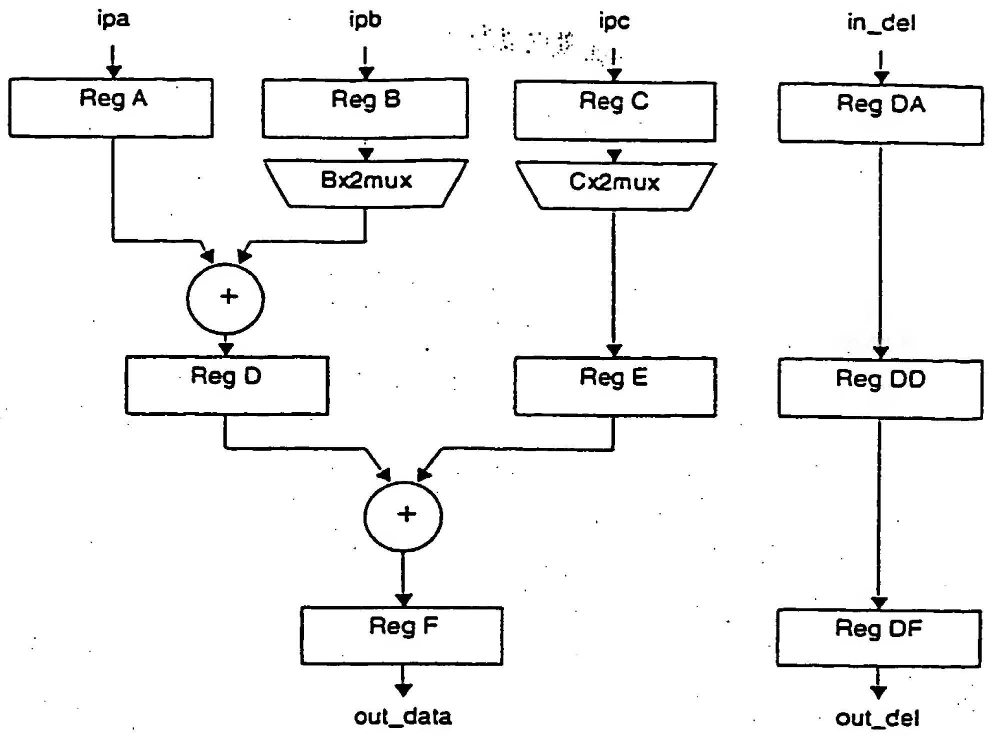


Figure 148

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Figure 149

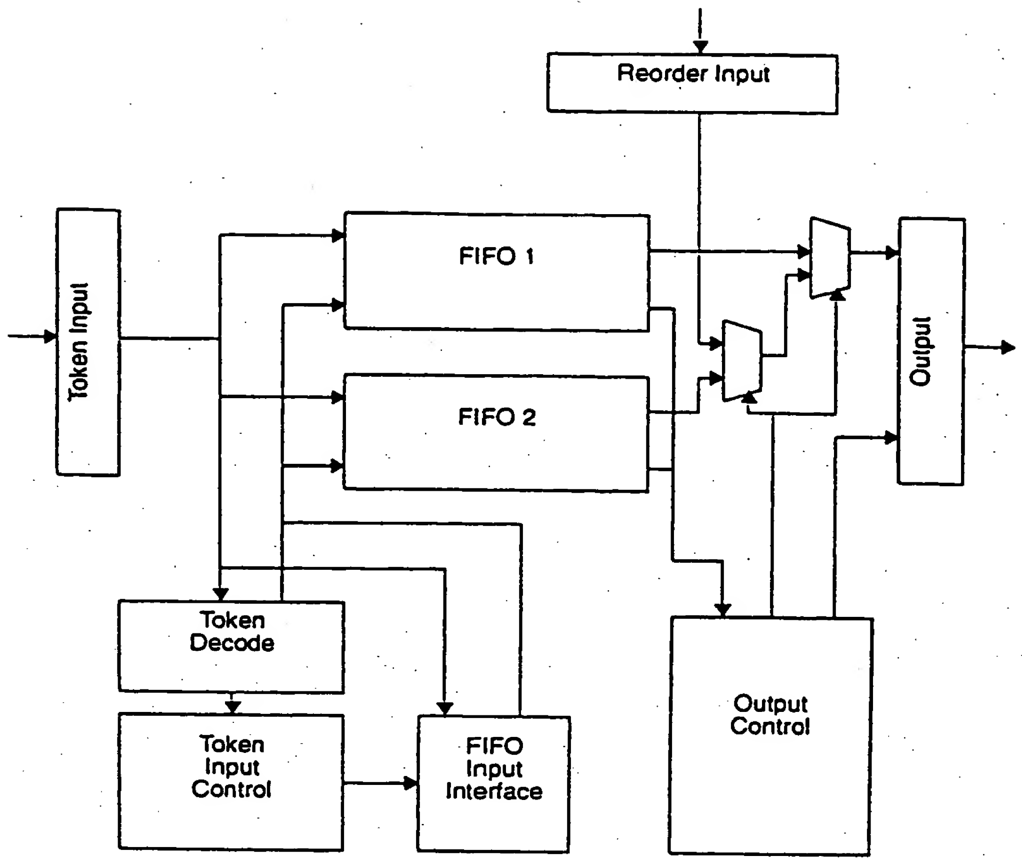

Reorder Input
Token Input
FIFO 1
FIFO 2
Output
Token Decode
Token Input Control
FIFO Input Interface
Output Control

Figure 150

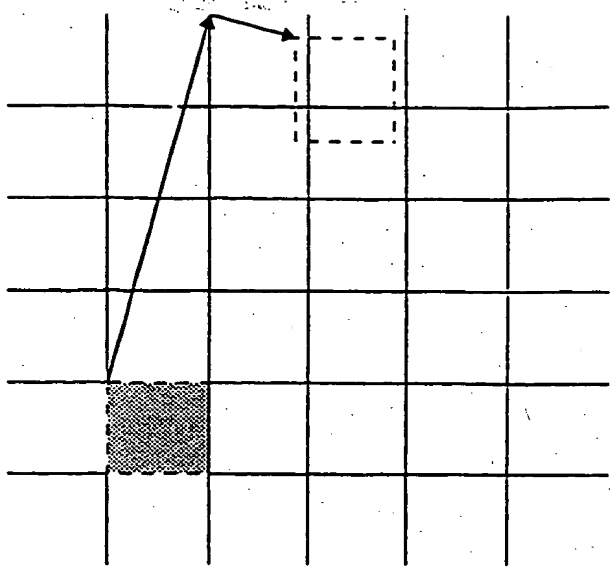


Figure 151

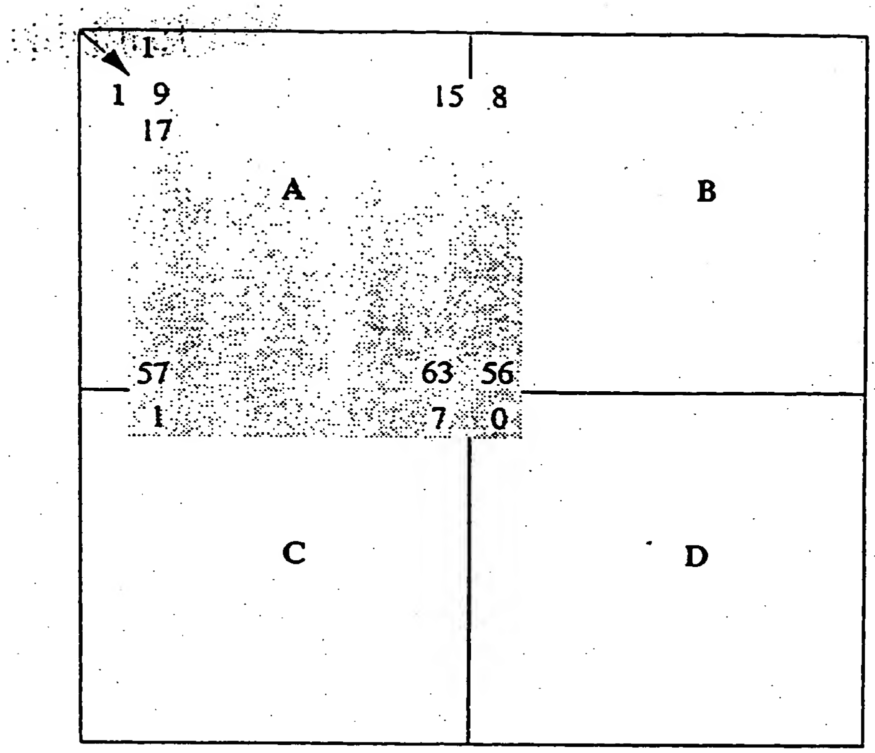


Figure 152

Read Cycle

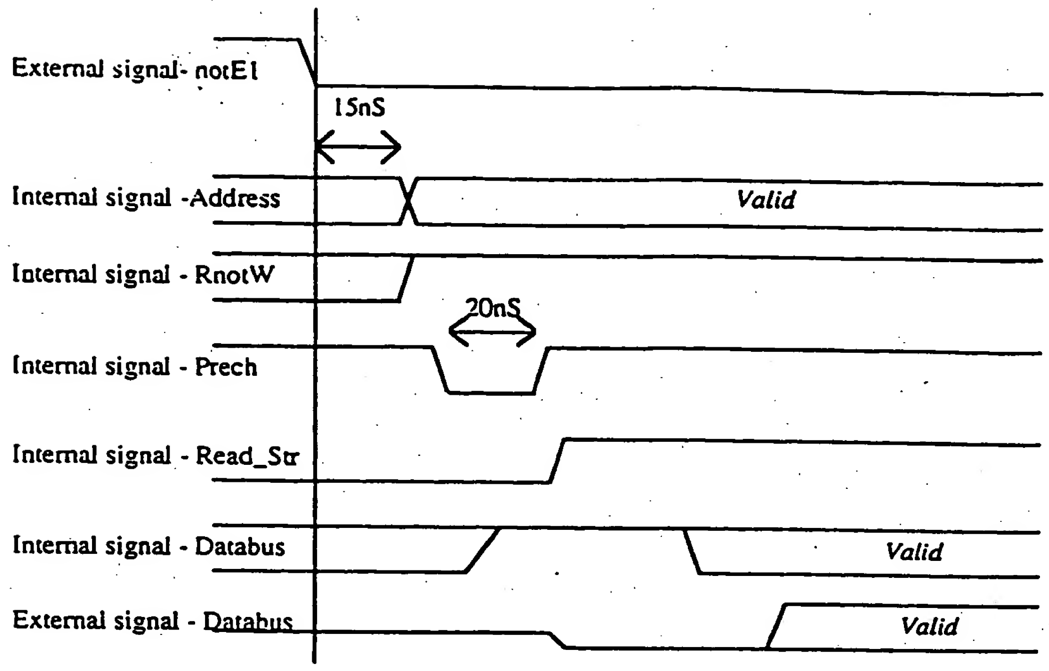


Figure 153

Write Cycle

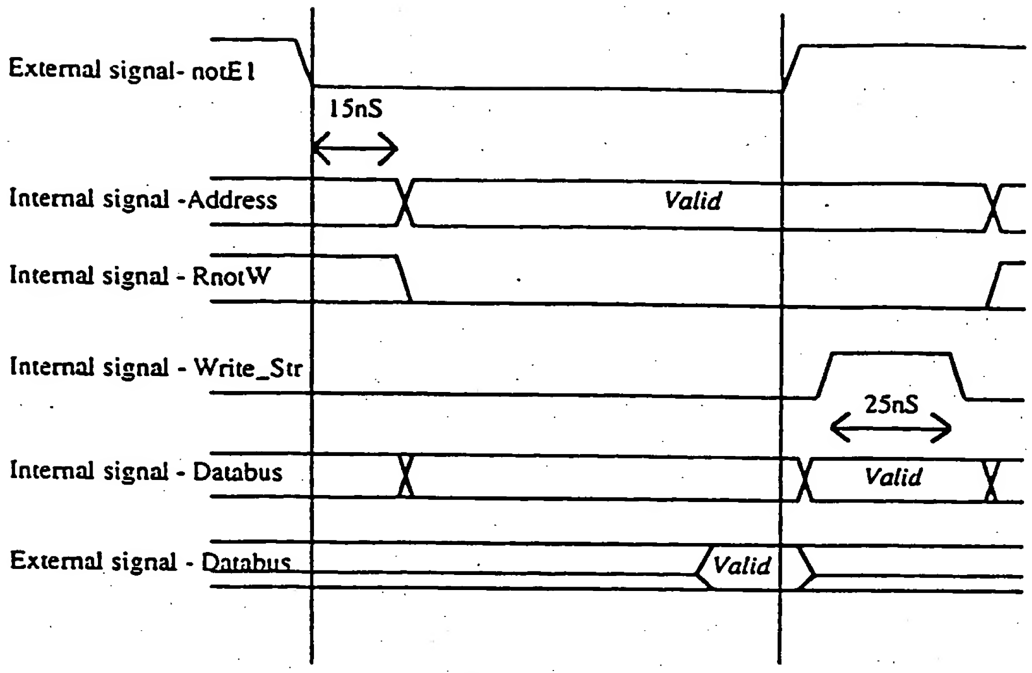


Figure 154

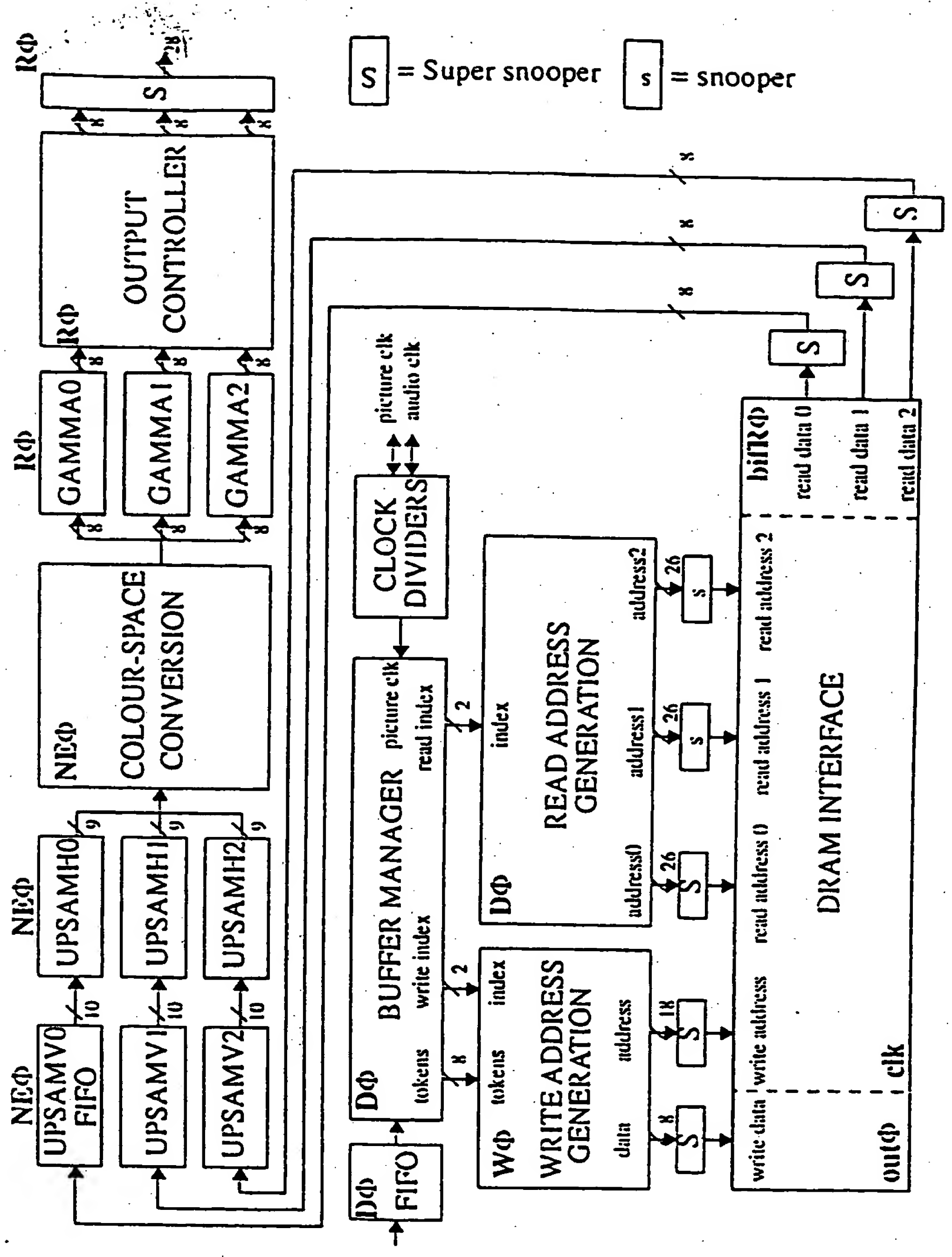


Figure 155

114

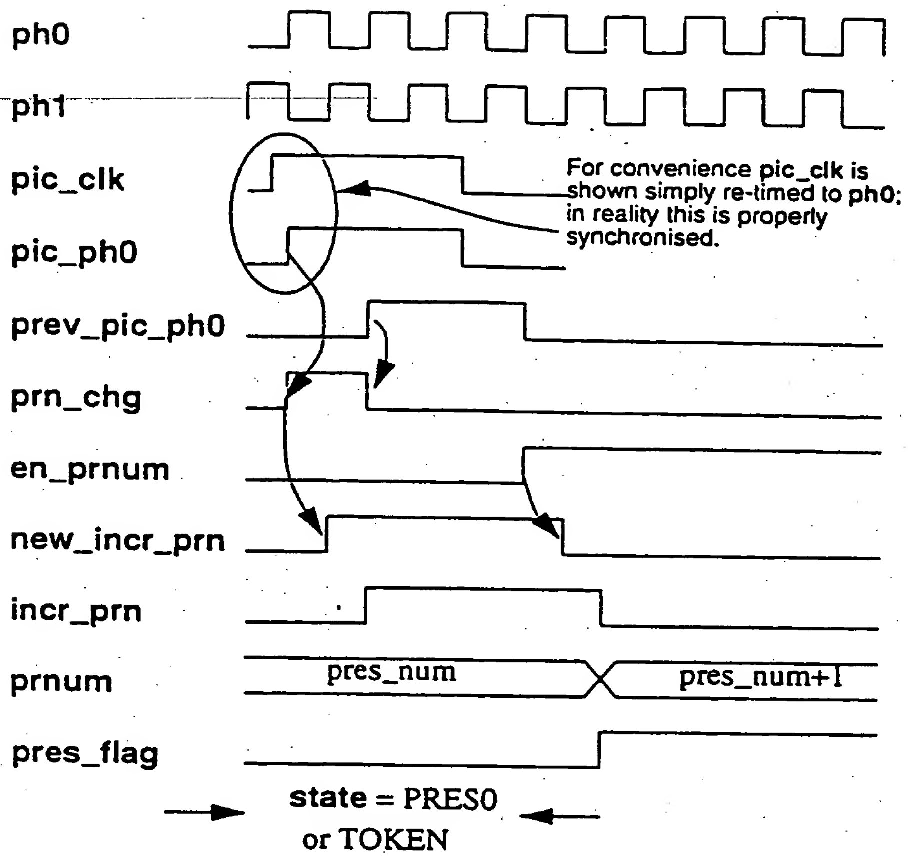

ph0
ph1
pic_clk
pic_ph0
prev_pic_ph0
prn_chg
en_prnum
new_incr_prn
incr_prn
prnum
pres_flag
For convenience pic_clk is shown simply re-timed to ph0: in reality this is properly synchronised.
pres_num
pres_num+1
state = PRES0 or TOKEN

Figure 156

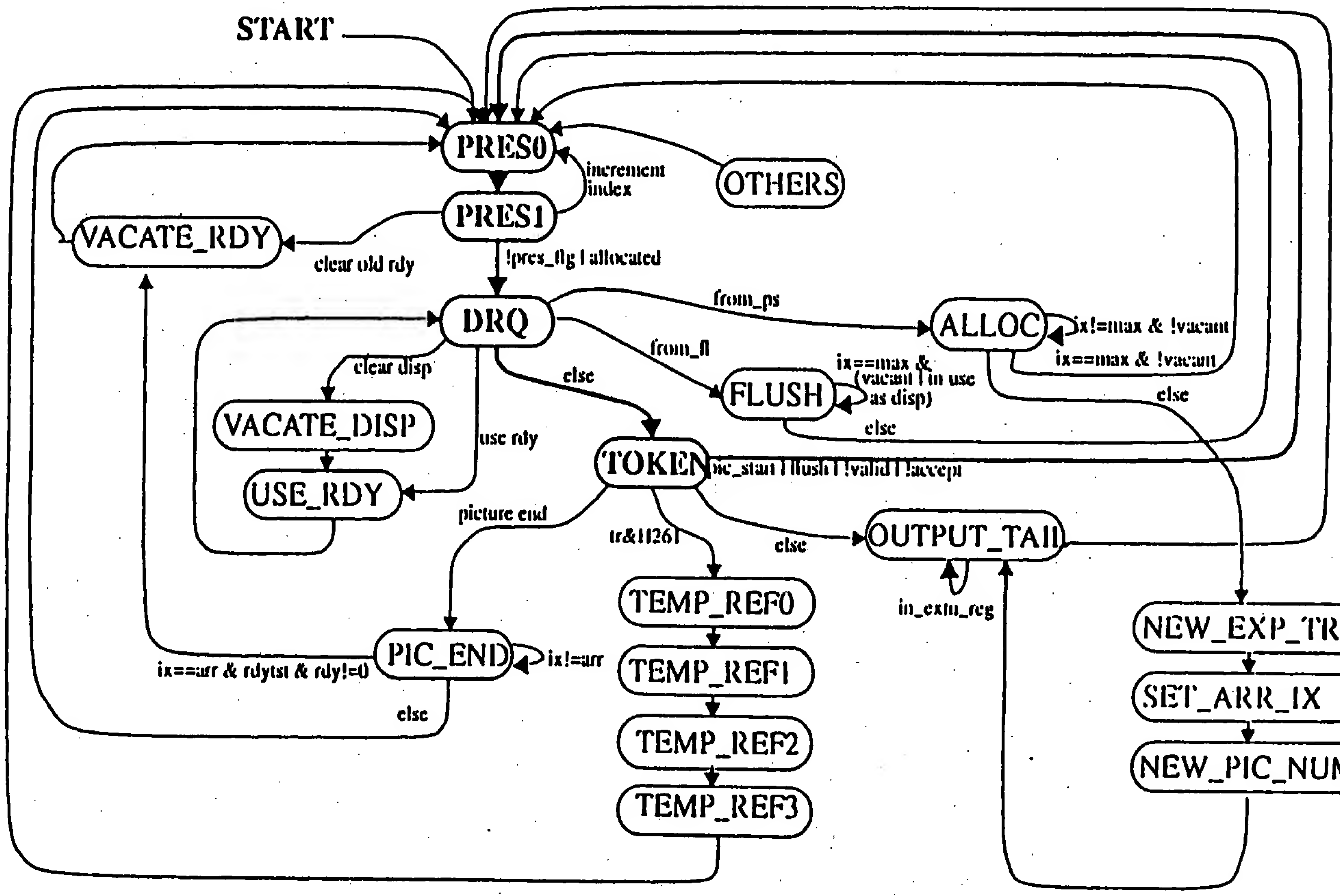

START
PRES0
PRES1
increment index
OTHERS
VACATE_RDY
clear old rdy
!pres_flg | allocated
DRQ
from_ps
ALLOC
ix!=max & !vacant
ix==max & !vacant
else
from_fl
FLUSH
ix==max & (vacant | in use as disp)
else
clear disp
VACATE_DISP
USE_RDY
use rdy
else
TOKEN
pic_start | flush | !valid | !accept
picture end
tr&11261
else
OUTPUT_TAIL
in_extn_reg
TEMP_REF0
TEMP_REF1
TEMP_REF2
TEMP_REF3
NEW_EXP_TR
SET_ARR_IX
NEW_PIC_NUM
PIC_END
ix!=arr
ix==arr & rdytst & rdy!=0
else

Figure 157

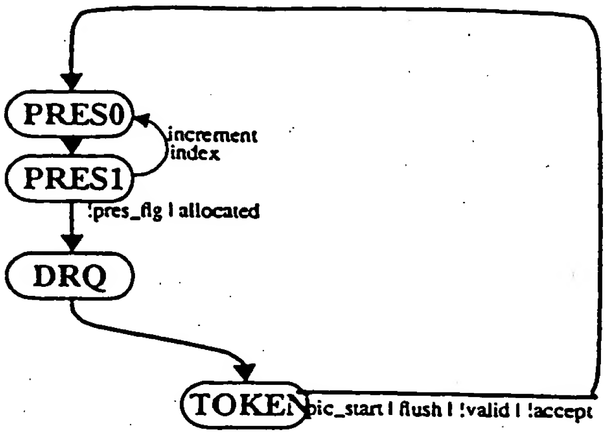
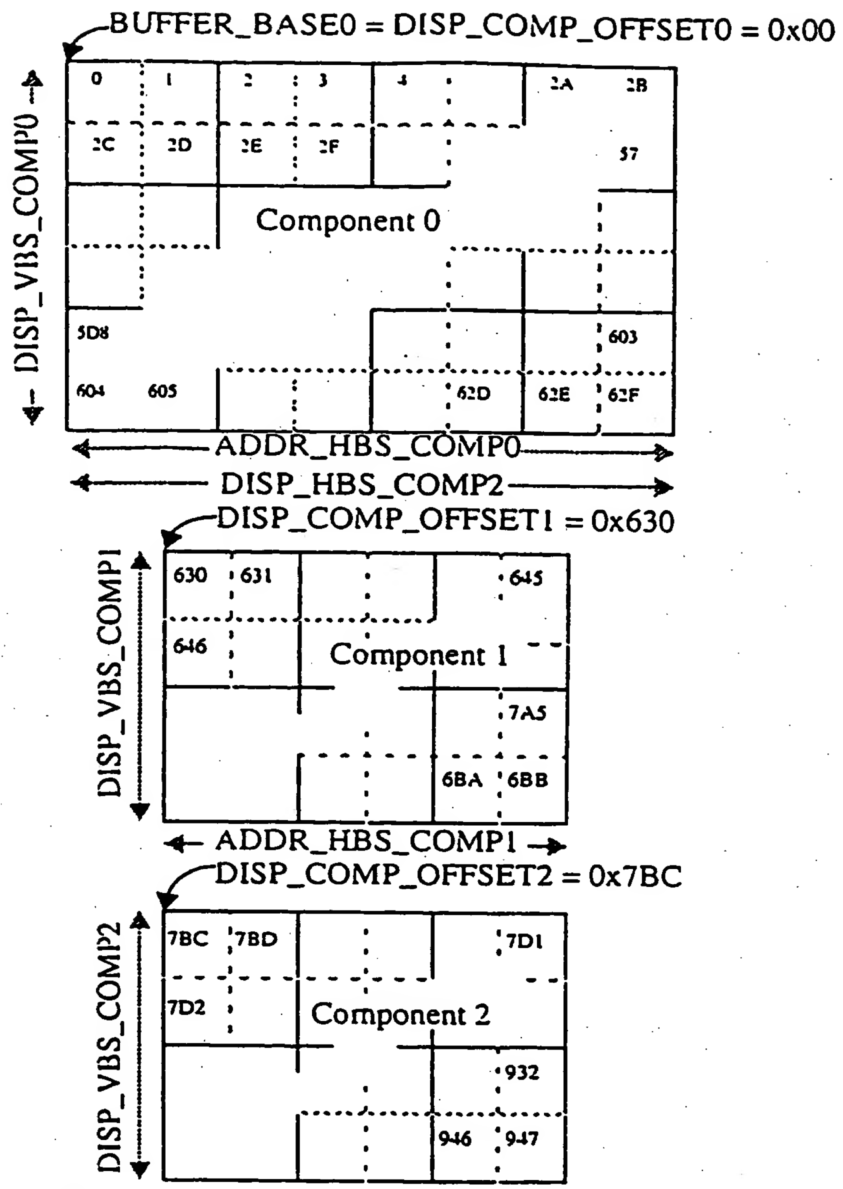

PRES0
increment index
PRES1
!pres_flg | allocated
DRQ
TOKEN
pic_start | flush | !valid | !accept

Figure 158



BUFFER_BASE0 = DISP_COMP_OFFSET0 = 0x00
0
1
2
3
4
2A
2B
2C
2D
2E
2F
57
Component 0
5D8
603
604
605
62D
62E
62F
DISP_VBS_COMP0
ADDR_HBS_COMP0
DISP_HBS_COMP2
DISP_COMP_OFFSET1 = 0x630
630
631
645
646
Component 1
7A5
6BA
6BB
DISP_VBS_COMP1
ADDR_HBS_COMP1
DISP_COMP_OFFSET2 = 0x7BC
7BC
7BD
7D1
7D2
Component 2
932
946
947
DISP_VBS_COMP2

Figure 159

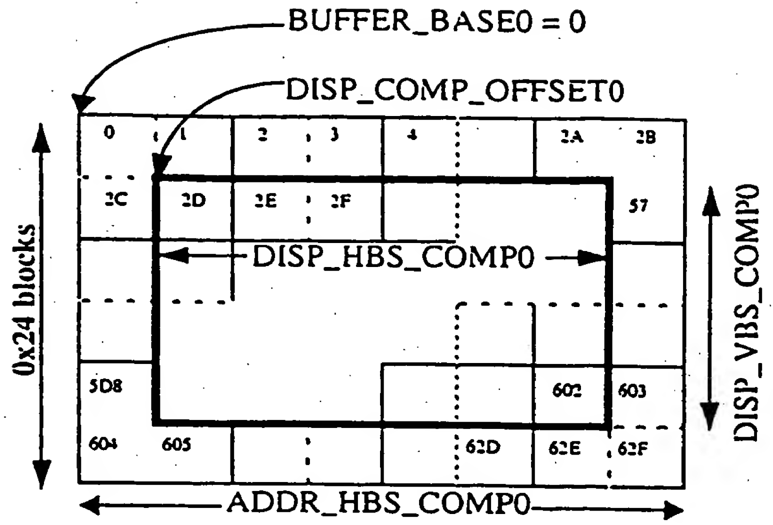

BUFFER_BASE0 = 0
DISP_COMP_OFFSET0
0
1
2
3
4
2A
2B
2C
2D
2E
2F
57
DISP_HBS_COMP0
0x24 blocks
DISP_VBS_COMP0
5D8
602
603
604
605
62D
62E
62F
ADDR_HBS_COMP0

Figure 160

Buffer offset 0x00:

Component0 offset 0x000 + ...

00	01	02	03	04	05	06	07	08	09	0A	0B
0C	0D	0E	0F	10	11	12	13	14	15	16	17
18	19	1A	1B	1C	1D	1E	1F	20	21	22	23
24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B
3C	3D	3E	3F	40	41	42	43	44	45	46	47
48	49	4A	4B	4C	4D	4E	4F	50	51	52	53
54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B
6C	6D	6E	6F	70	71	72	73	74	75	76	77
78	79	7A	7B	7C	7D	7E	7F	80	81	82	83
84	85	86	87	88	89	8A	8B	8C	8D	8E	8F

Component1 offset 0x100 + ...

00	01	02	03	04	05
06	07	08	09	0A	0B
0C	0D	0E	0F	10	11
12	13	14	15	16	17
18	19	1A	1B	1C	1D
1E	1F	20	21	22	23

Component2 offset 0x200 + ...

00	01	02	03	04	05
06	07	08	09	0A	0B
0C	0D	0E	0F	10	11
12	13	14	15	16	17
18	19	1A	1B	1C	1D
1E	1F	20	21	22	23

Figure 161

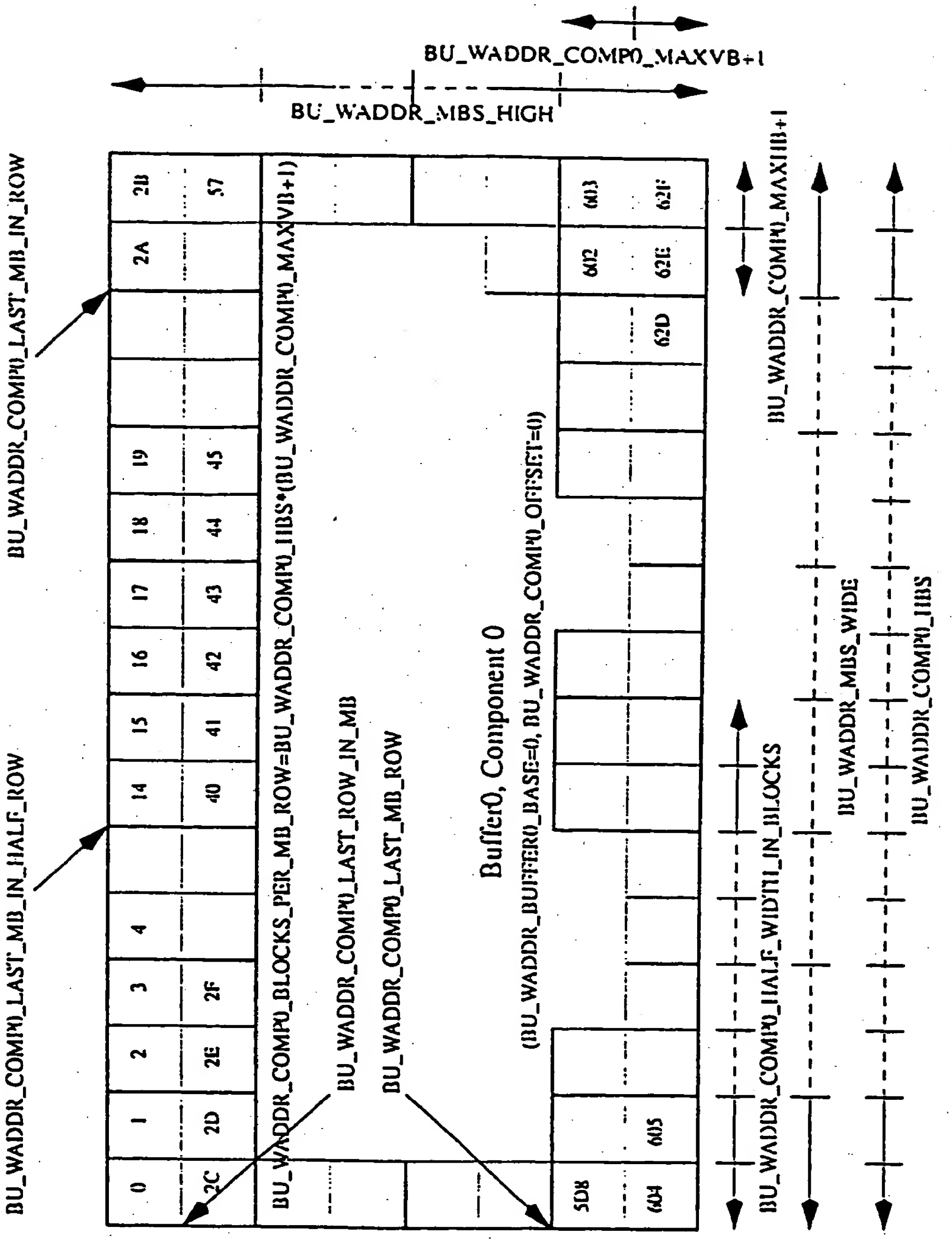
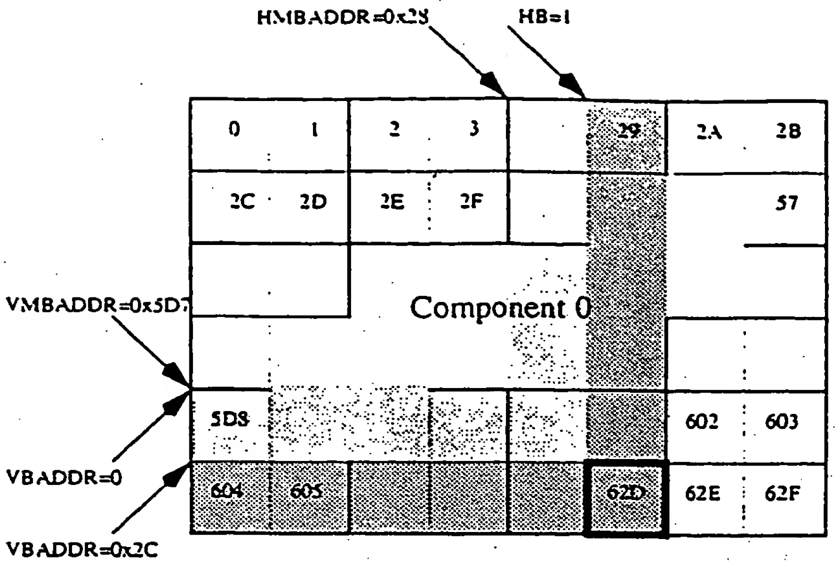
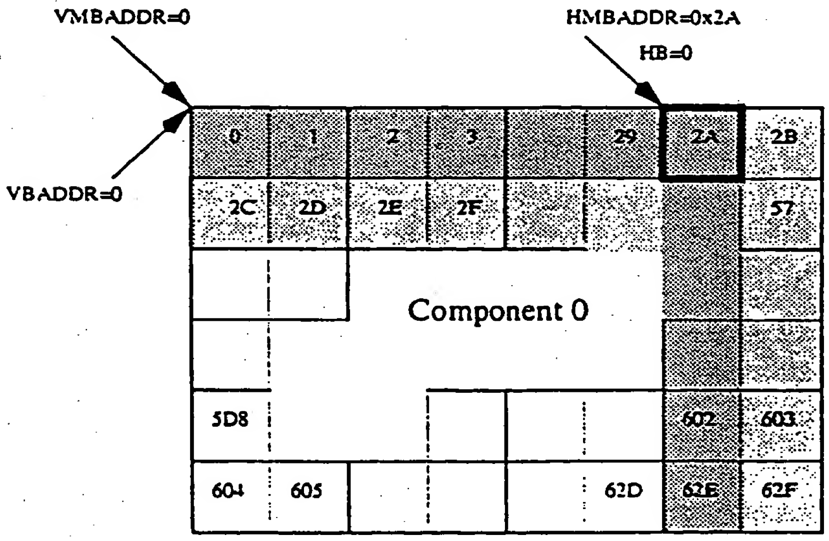

BU_WADDR_COMP0_MAXVB+1
BU_WADDR_MBS_HIGH
BU_WADDR_COMP0_LAST_MB_IN_ROW
BU_WADDR_COMP0_LAST_MB_IN_HALF_ROW
BU_WADDR_COMP0_BLOCKS_PER_MB_ROW=BU_WADDR_COMP0_HBS*(BU_WADDR_COMP0_MAXVB+1)
BU_WADDR_COMP0_LAST_ROW_IN_MB
BU_WADDR_COMP0_LAST_MB_ROW
Buffer0, Component 0
(BU_WADDR_BUFFER0_BASE=0, BU_WADDR_COMP0_OFFSET=0)
BU_WADDR_COMP0_MAXHB+1
BU_WADDR_COMP0_HALF_WIDTH_IN_BLOCKS
BU_WADDR_MBS_WIDE
BU_WADDR_COMP0_HBS
0
1
2
3
4
14
15
16
17
18
19
2A
2B
2C
2D
2E
2F
40
41
42
43
44
45
57
5D8
604
605
62D
602
62E
603
62F

Figure 162

121



block address = 0 + 0 + 0x5D8 + 0x28 + 0x2C + 1 = 0x62D



block address = 0 + 0 + 0 + 0x2A + 0 + 0 = 0x2A

Figure 163

127

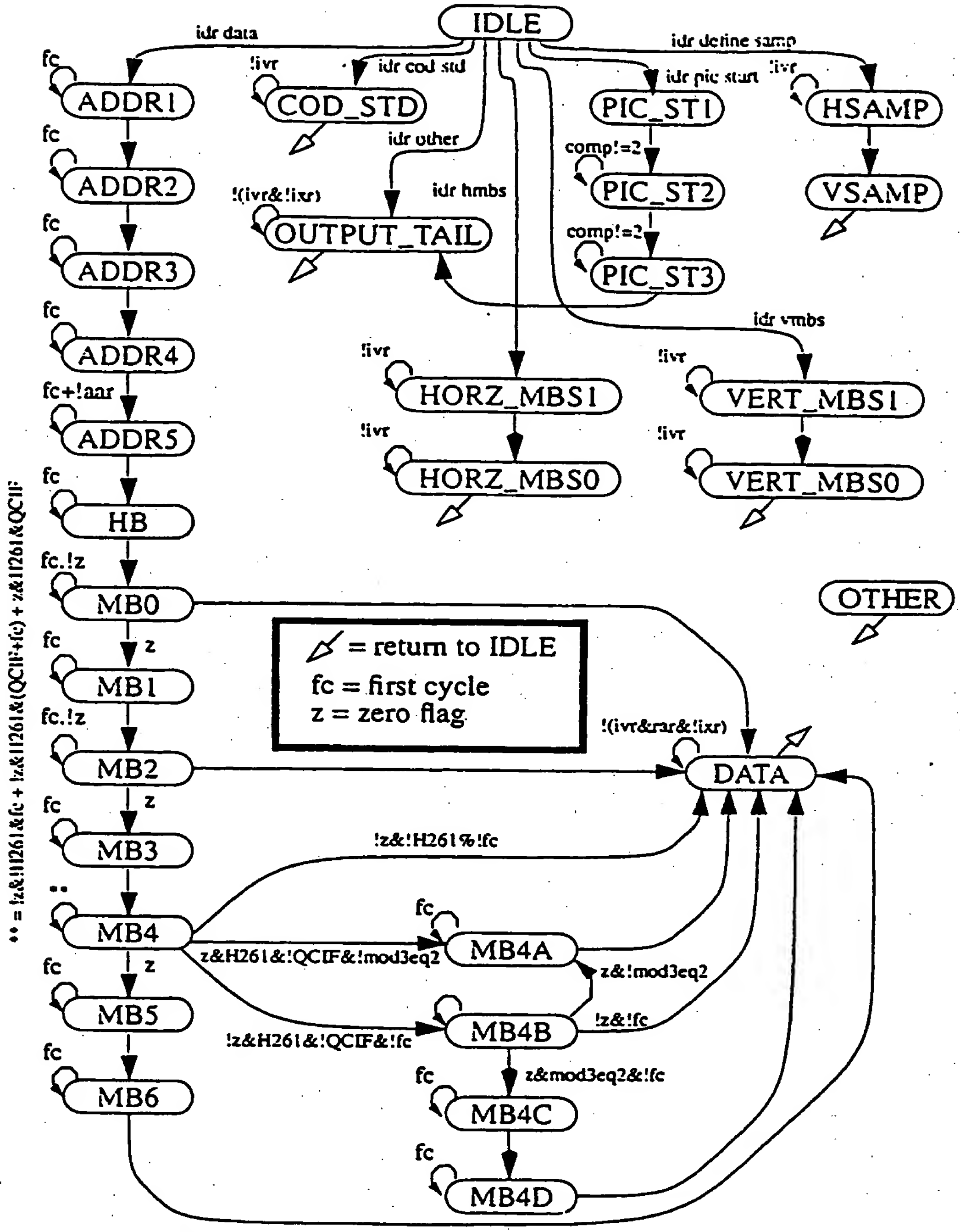


Figure 164

123

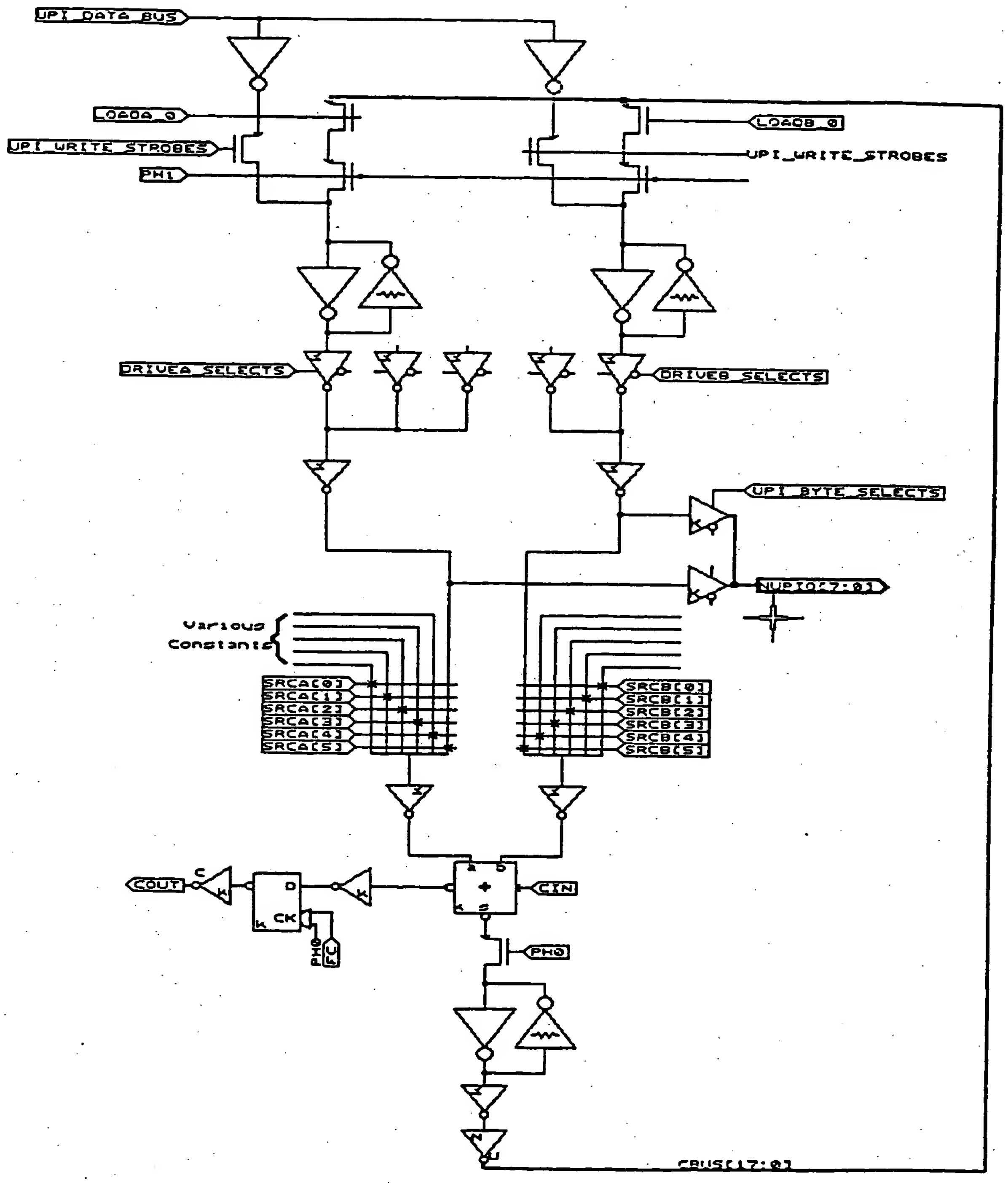

UPI_DATA_BUS
LOADA_0
UPI_WRITE_STROBES
PH1
LOADB_0
UPI_WRITE_STROBES
DRIVEA_SELECTS
DRIVEB_SELECTS
UPI_BYTE_SELECTS
NUPIO[7:0]
Various Constants
SRCA[0]
SRCA[1]
SRCA[2]
SRCA[3]
SRCA[4]
SRCA[5]
SRCB[0]
SRCB[1]
SRCB[2]
SRCB[3]
SRCB[4]
SRCB[5]
COUT
D
CK
PH0
CIN
PH0
CBUS[17:0]

Figure 165

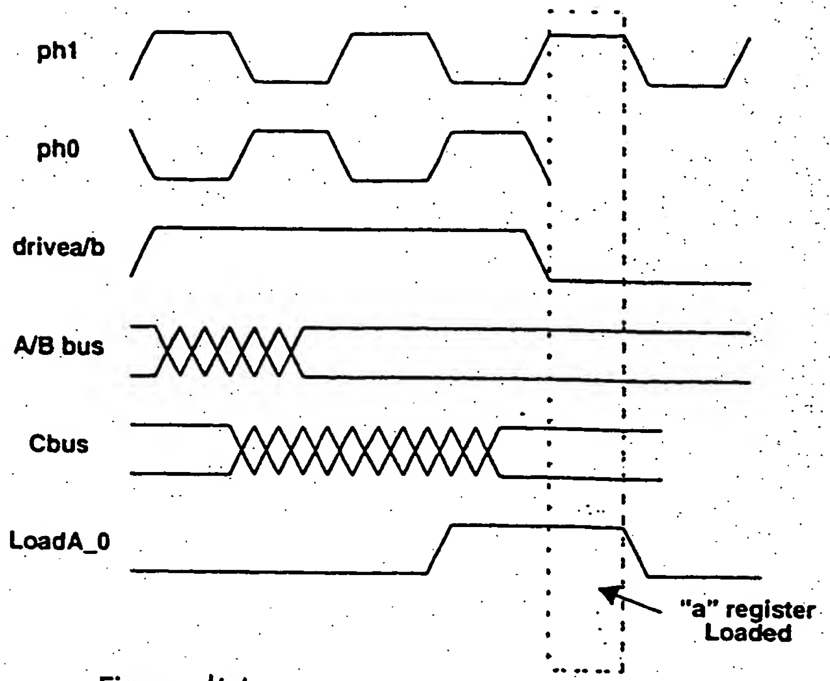


Figure 166

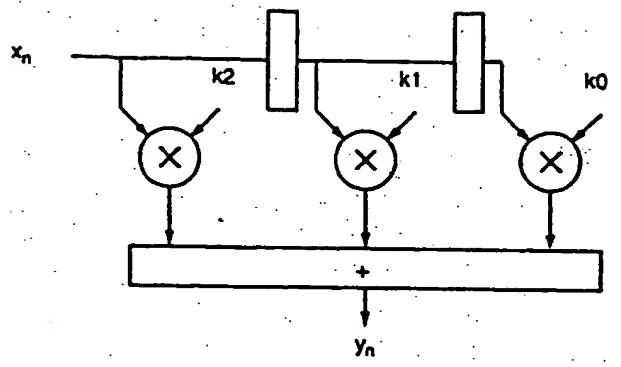


Figure 167

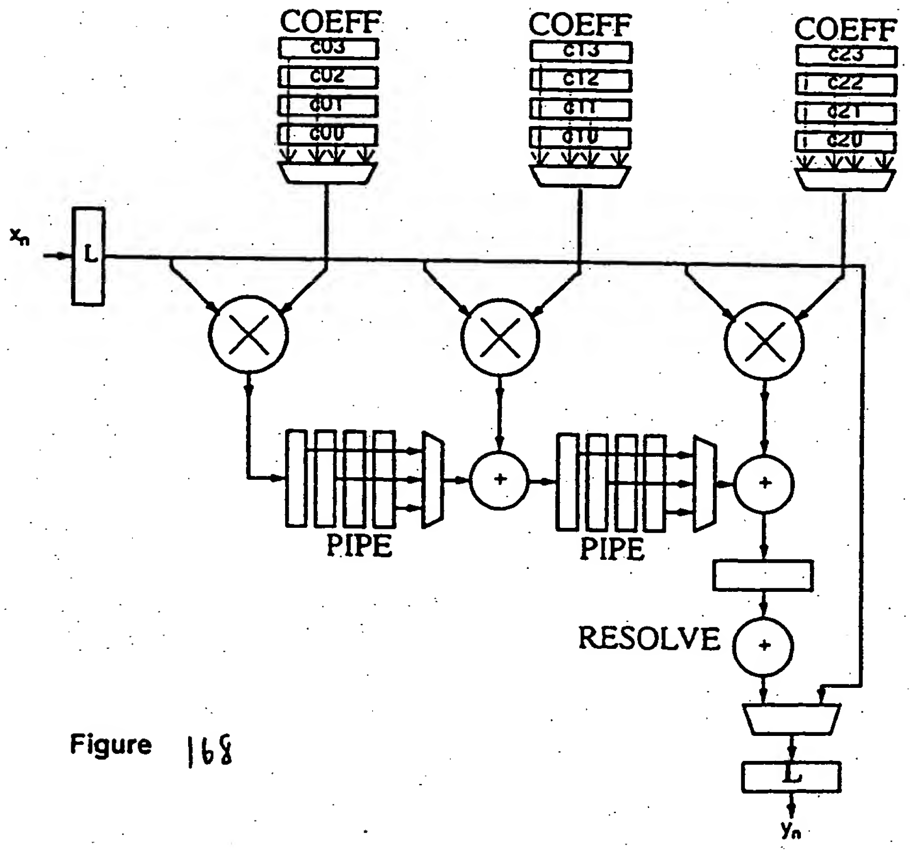


Figure 168

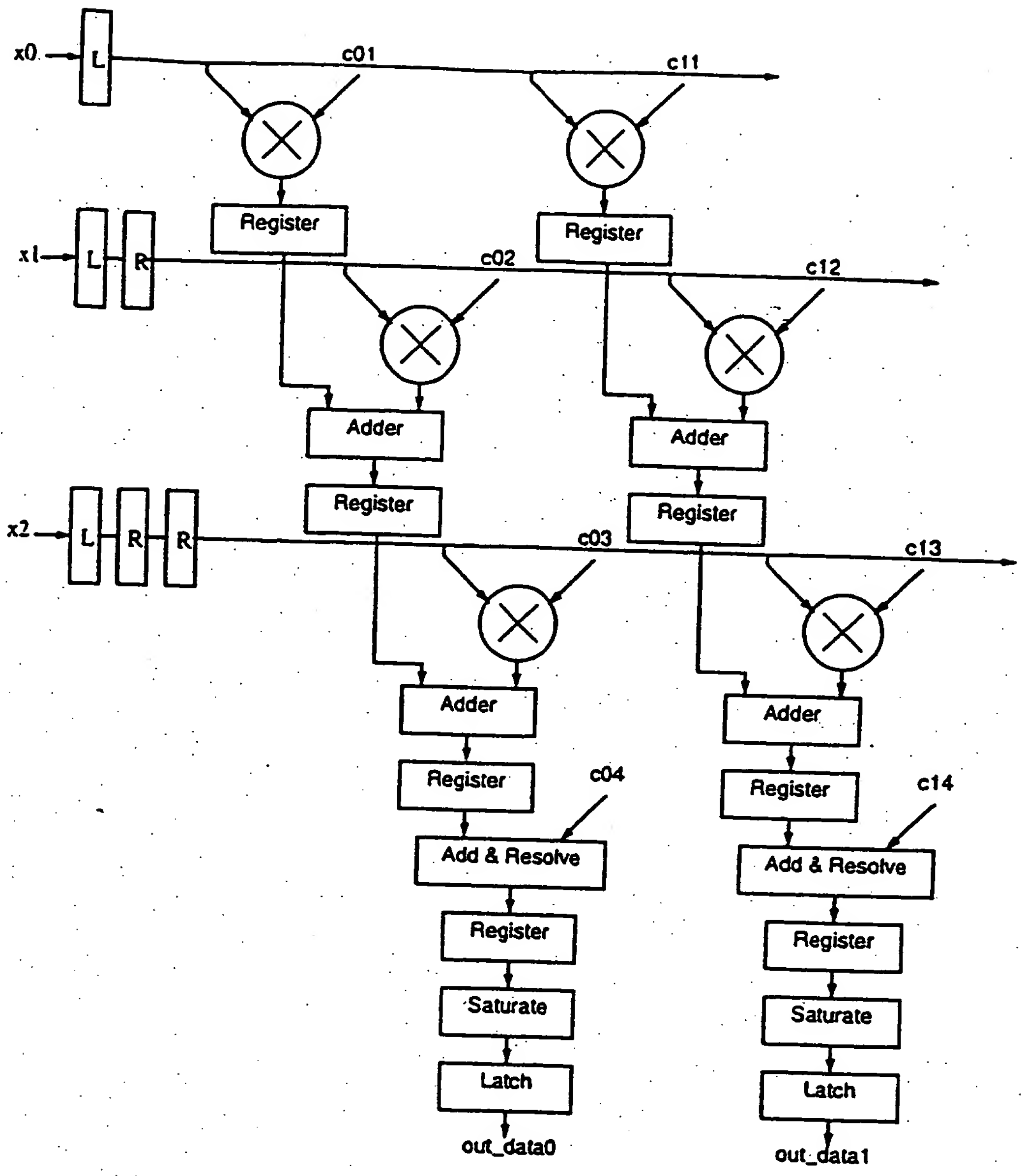

x0
L
c01
c11
Register
Register
x1
L
R
c02
c12
Adder
Adder
Register
Register
x2
L
R
R
c03
c13
Adder
Adder
Register
Register
c04
c14
Add & Resolve
Add & Resolve
Register
Register
Saturate
Saturate
Latch
Latch
out_data0
out_data1

Figure 169